MES Help Documentation

Table of Contents

1 Welcome to MES space

Manufacturing Execution Systems (MES) software is used to supervise and track work on a plant floor. Our MES Modules offer vertical software solutions built on the power of the Ignition platform. It provides functions such as resource management, detailed scheduling, dispatching, production analysis and downtime management, product tracking and genealogy, monitoring recipes, quality management, variance tracking, real-time predictive analysis, and more. MES is the layer between SCADA and ERP systems.

Sepasoft Support

- Open a ticket with our support team
- Email us
- Call us toll free at (800) 207-5506, or at (916) 939-1684, if you're calling from outside the U.S.
- Or learn all about getting trained at our Learning portal

2 Documentation Organization

This comprehensive resource gives a broad overview of our MES Software, the basic instructions about its configuration, along with information on design and coding. It is intended for developers and those looking for detailed technical information. Every effort has been made to ensure that this document is an accurate representation of the functionality of MES. The deployment instructions included can help you get started with MES development as soon as you download. All of the MES modules are built on the Ignition platform.

The document repository is broken into two main sections....

2.1 Knowledge Base Articles

Here you'll find useful articles describing how to do certain things. Generally the articles will contain an overview, steps, screenshots and code blocks that you can use to extend the MES framework. The articles have generally come from real world applications where modifications to the normal MES implementation was required. There articles show how to modify the standard behavior through custom scripting or use of built-in functions.

2.2 User Manuals

This section contains information on 'Getting Started', 'Theory of Operation' and has the reference manual for MES components, ISA-95 objects, functions and scripting.

Here are some common links...

- Download and Install MES
- What is MES?
- Quick Tour
- Architecture and Modules

2.3 Using the Search Function

You can use this search box  at the top of the page to quickly find what you're looking for. Here are some tips on how best to use it to find what you're after....

- If you're searching for a specific function i.e. '**system.mes.object.createFilter()**', then typing it in exactly will get you to where you want to be. Typing in '**mes.object.createFilter()**' will not. Typing in '**mes object createfilter()**' will find it based on the keywords **mes object createFilter**.

Let's get started!

We recommend using the online version of our user manual rather than downloading it's PDF since we are updating it daily. If you chose to use the PDF, click on the  icon in the sidebar.

3 MES Products

Click on the logo to direct you to corresponding module description. Also see the Sales Info button for each modules.

3.1 OEE Downtime

Sales Info

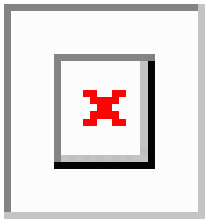
3.2 Track and Trace

Sales Info

3.3 SPC

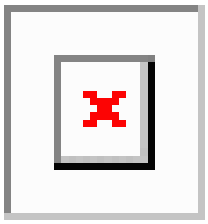
Sales Info

3.4 Recipe/Changeover



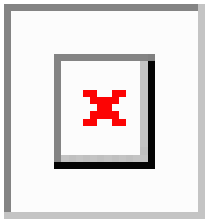
Sales Info

3.5 Instrument Interface



Sales Info

3.6 Web Services



Sales Info

4 Getting Started

4.1 New to MES?

Lets get started by running MES on your computer...

Our free trial lets you evaluate the full version of the software without requiring any license.

Download and Install MES

4.2 Basics in MES

Read these topics and watch the Quick Tour...

- What is MES?
- Quick Tour
- Architecture and Modules

4.3 A Simple Workflow in MES

Step 1. Database Connection

Step 2. Configuring MES Databases

Step 3. Installing the Production Simulator

Step 4. Production Model Configuration

4.4 Sepasoft Resources

Online Help

You can access Online Help from the Ignition Gateway and the Designer. To access online Help from the Gateway, go to the Configure section and select **System > User Manual**.

To access it from the Designer, click the **F1** key or select **Help > Help** from the top menus.

MES Training

Explore a broad overview of Sepasoft's MES modules . By taking a training course taught by one of our experts, you'll learn key skills and how to apply them to real-world projects .

Get an overview of OEE, TEEP, downtime, SPC, recipe management, and track and trace. The course covers installation, production model configuration, analysis, modifying user screens and reports. In-Person Training is also available. For more information go to MES Training.

Learn

You can watch the MES training videos, test your knowledge, and participate in our new credential program.

When you need information on any given feature, you can read about the feature here in the User Manual, then go to the same section in the Video Library course list to watch the related video.

When there is a corresponding video for a feature description, you will see a video link as follows:

Watch the Video

MES Video

Support Team

For one-on-one help from our support team, go to the Support homepage and submit a Ticket. One of our Support engineers will follow up with you quickly. You can reach us during business hours 8am-5pm PST at 1-800-207-5506 . Support charges may apply. 24-hour support is also available, at an additional fee.

E-mail support is available at support@sepasoft.com .

Knowledge Base

Search and view all the Knowledge Base articles for troubleshooting, known problems, and workarounds.

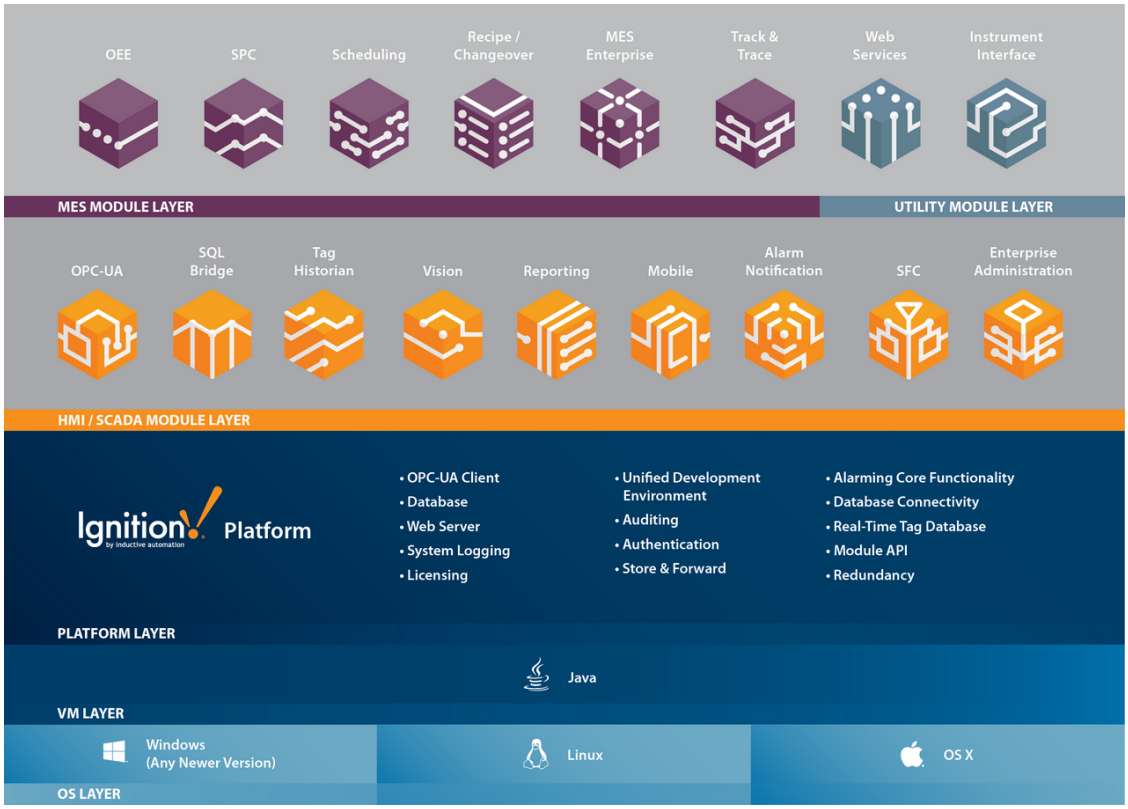
Other Resources

- White Papers and Articles
- Webinars on Demand
- Tip Sheets

5 Installation Guide

This is a guide to installing the MES modules. For more information on the modules themselves, please see the Module section.

> The MES Modules sit on top of the Ignition platform by Inductive Automation. If you do not have Ignition already installed, please download the current version of Ignition from the Inductive Automation web-site downloads page.

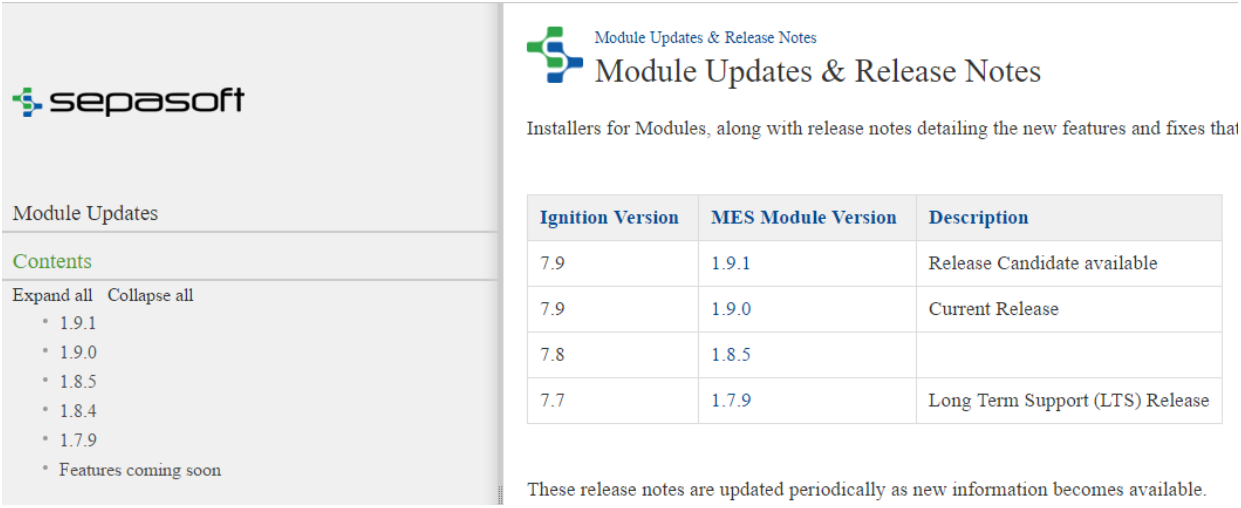


5.1 Installing the MES Modules

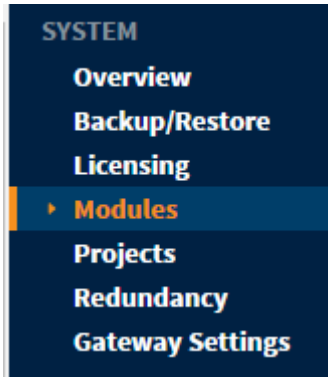
Download the MES modules by going to the Sepasoft downloads page. Under the modules section, you can find the following modules.

- Track & Trace
- OEE Downtime
- SPC

- Recipe & Changeover
- Instrument Interface
- Web Services
- Barcode Scanner
- MES Enterprise

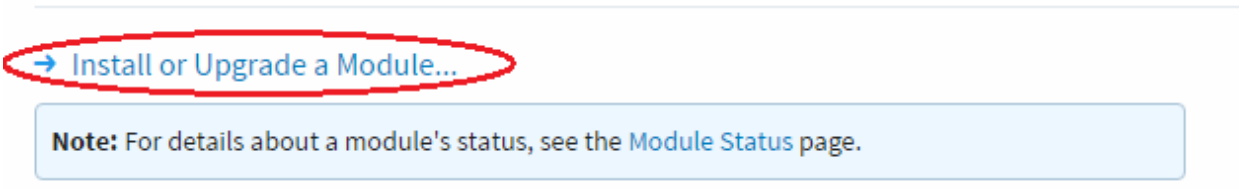


Once you have the modules downloaded, log in to the Ignition Gateway webpage and select **Configuration > Modules** from the menu.

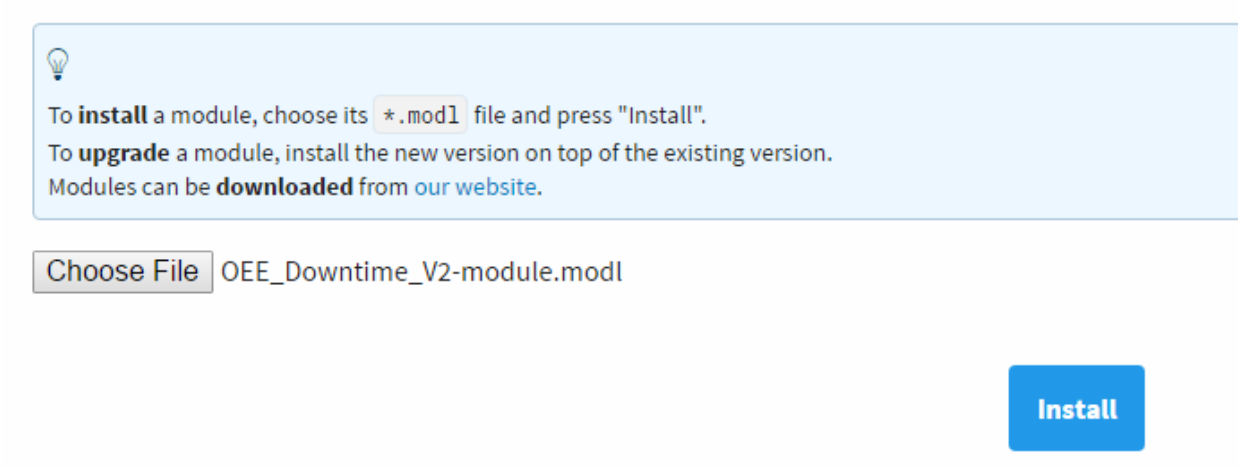


5.1.1 Install or Upgrade Module ?

Install the modules (.modl files) one at a time by scrolling down to the bottom of the list and clicking on **Install or Upgrade a Module....**



Browse for your module and click **Install**.



The MES Modules should now be installed.

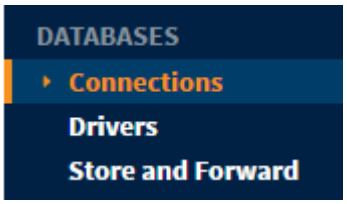
5.2 Database Connection

In order to use the MES Modules you have to connect the Ignition Gateway to a SQL database. The MES Modules simply need a blank database and they will create all of the necessary tables and maintain the data.

MES data is stored in databases external to Ignition. These database(s) are setup in the gateway configuration section by selecting the Databases> Connections section from the left-hand configuration menu. See the Ignition documentation for more information on setting up a database connection.

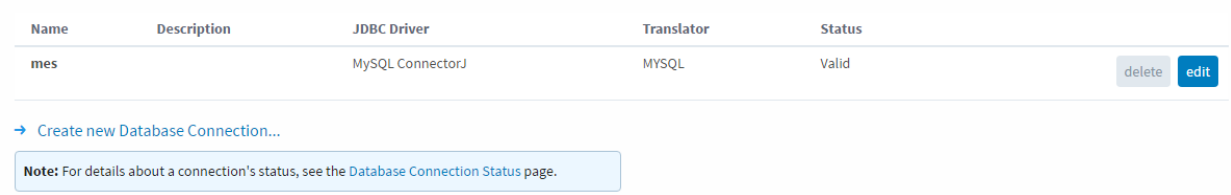
Currently, the MES Suite supports MySQL, SQL Server, Oracle and PostgreSQL.

You can setup a database connection in the Ignition Gateway configuration section. Log in to configuration area and select **Databases > Connections** from the menu.

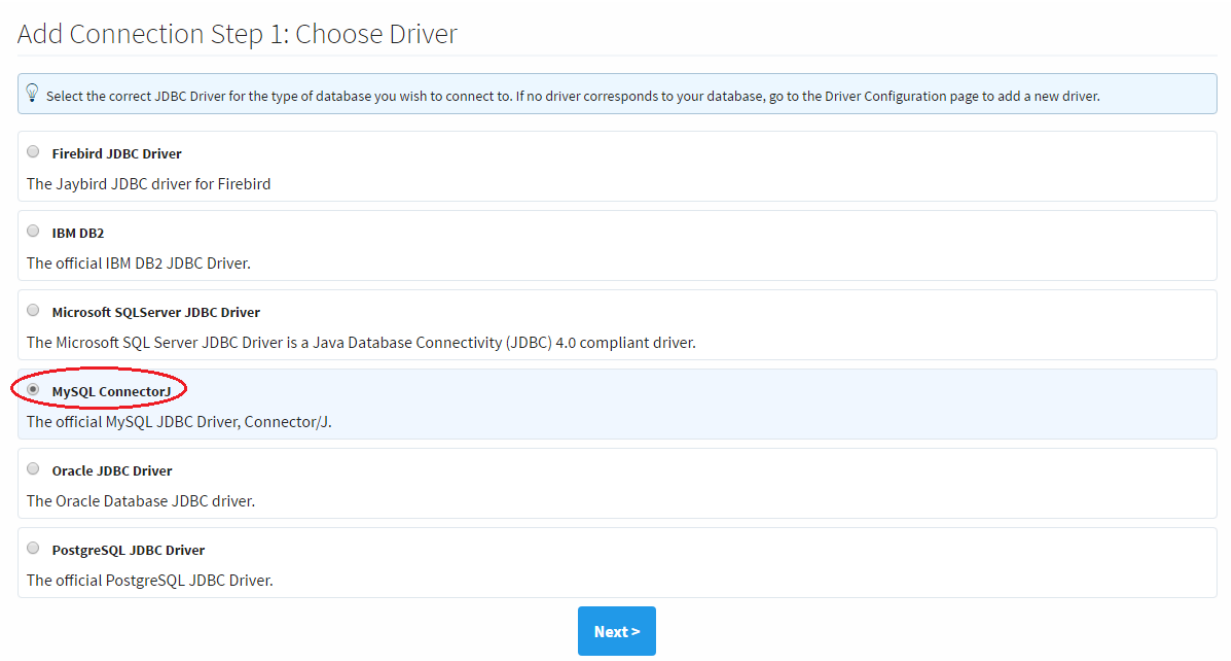


5.2.1 Create a new Database Connection

Click on the **Create new Database Connection** to add a new connection.



Choose the driver for the database you plan on using. In the example, we have chosen the **MySQL ConnectorJ**.



Give the connection a name, specify the connect URL, and the credentials to connect. Take a look at this example:

Name: MES

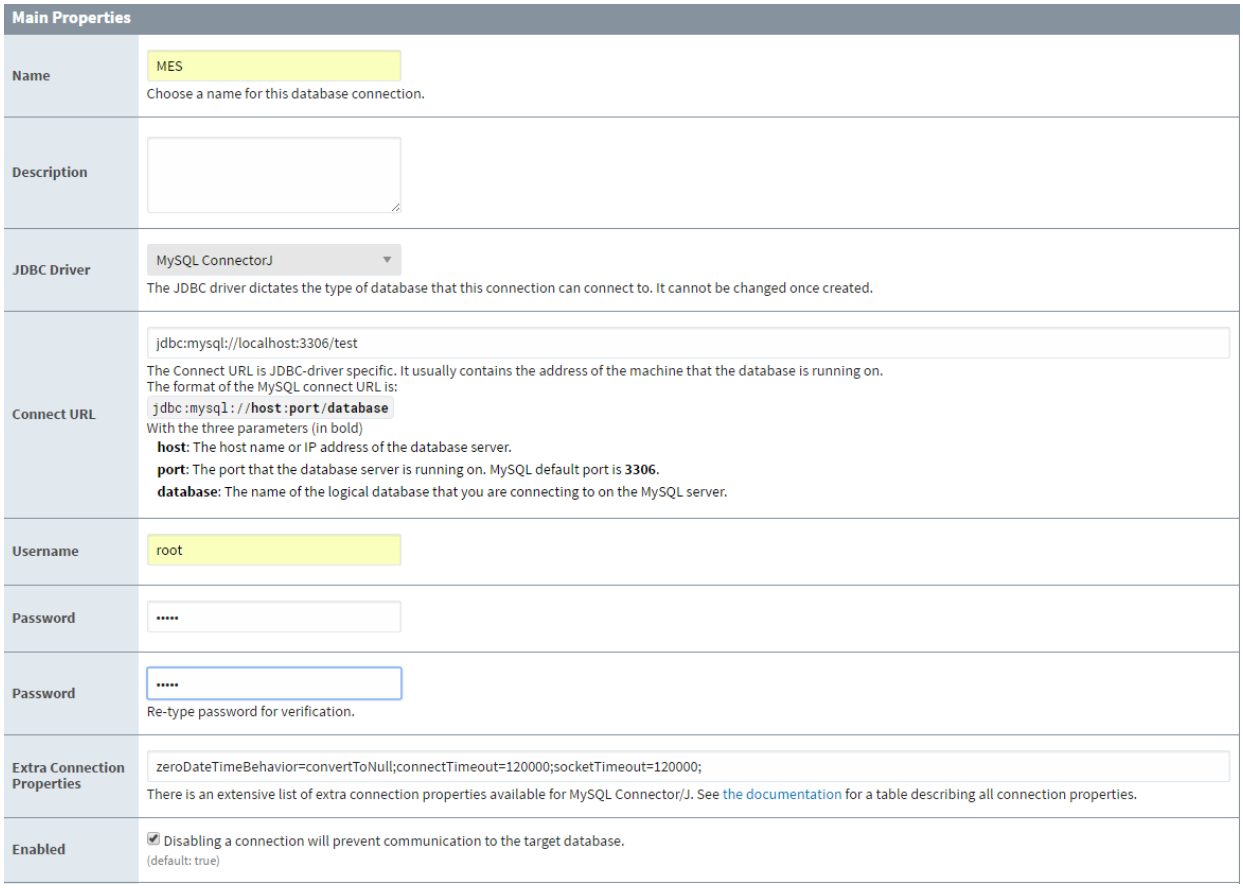
Connect URL: jdbc: mysql://localhost:3306/test

Username: root

Password: mysql

Once you have the information entered in click on the **Create New Database Connection** button to finish creating the connection.

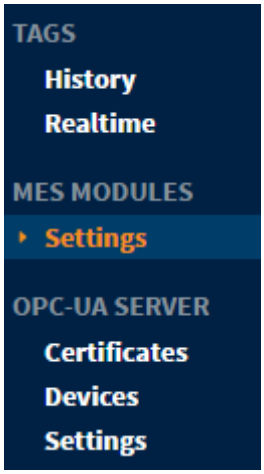
Verify the connection is valid before continuing to the next section. Now we have a database to store the MES data in.



5.3 MES Module Settings

The Track and Trace, OEE Downtime, Scheduling, SPC and Recipe modules store data in the SQL database defined in this section. All MES database tables and indices will be created automatically in the selected database.

Because Ignition can be configured to multiple databases, the MES Module Settings configuration page is used to select which databases to store the data. To change the MES module settings, go to the configuration section in the gateway and select the **MES Modules > Settings** section from the left-hand side configuration menu.



Once a database connection is created, and if only one database connection exists, then it will be automatically selected to be used by the MES modules. If more than one database connection exists, then the desired database connection can be selected to be used by the MES modules as shown below

> OEE, downtime and schedule data is stored in databases external to Ignition. Production and downtime data is stored in the runtime db during a production run. Production and downtime data is summarized and saved in the analysis db.

5.3.1 Authentication

User Source Profile

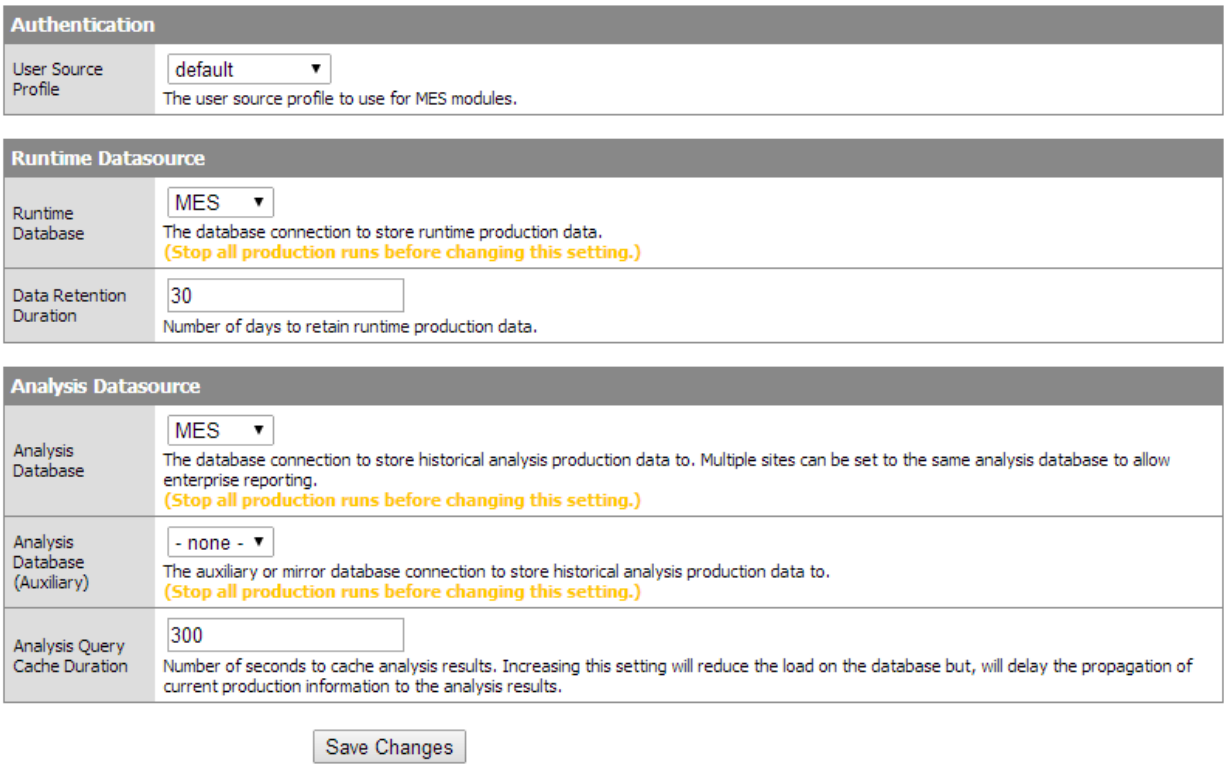
MES Modules derives MES Person objects from the any users that have been configured that have first or last name assigned. Since Ignition can have multiple User Source Profiles, the MES system must be configured to know which one to use. In the Authentication section, select the user source profile to have the MES system use.

5.3.2 Runtime Datasource

Runtime Database

The Runtime datasource points to the database where production and downtime data is stored during a production run. During a production run, data is logged every minute or partial minute if a downtime event occurs, so a large amount of data is stored in this database. Typically this database is local to the site running the MES Modules.

The runtime database is not used for the Track and Trace Module.



Data Retention Duration

This attribute can be used to limit how large this database comes by automatically purging this data after the defined time. Don't worry, the final OEE production data, downtime events and counts for each run are permanently stored in tables in the Analysis database. The OEE Time Chart component however, does use the data stored here.

Runtime database is not used by the Track and Trace Module.

5.3.3 Analysis Datasource

Analysis Database

The Analysis datasource points to the database where all final MES data is stored. For example, the OEE module stores summarized production and downtime data here that is used for analysis.

Analysis Database (Auxiliary)

The MES Modules will mirror the information that is written to the local analysis database to the Auxiliary database. For single site implementations, set this to **-none-**. When you have MES running on multiple servers, setting up the Auxiliary database connection will push MES data up to a central Database to allow enterprise analysis to be performed.

Analysis Query Cache Duration

This setting represents the number of seconds to cache analysis results. Analysis is used to compare the retrieved trace information for the trace graph, reporting and etc.

5.3.4 MES Object Cache

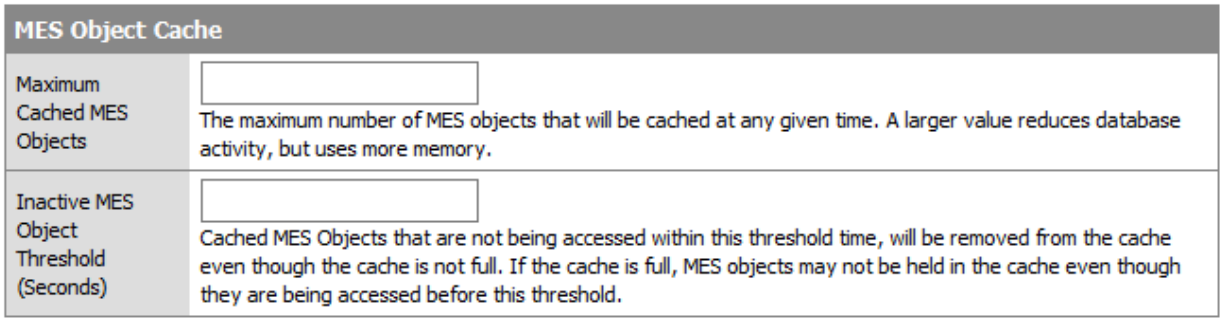
The MES Object Cache provides some configuration settings that can be used to affect database performance.

Maximum Cached MES Objects

The maximum number of MES objects that is cached at a specific time interval.

Inactive MES Object Threshold (Seconds)

The cached MES objects that are inactive within the specified threshold time would be removed even though the cache is not full.



Tip

In determining whether to keep the runtime and analysis databases separate, consideration should be given regarding the need to have separate database backup and archiving schemas. Recommendation here is to keep the runtime and analysis databases the same. This allows for simple SQL joins if any custom SQL queries are required in your application.

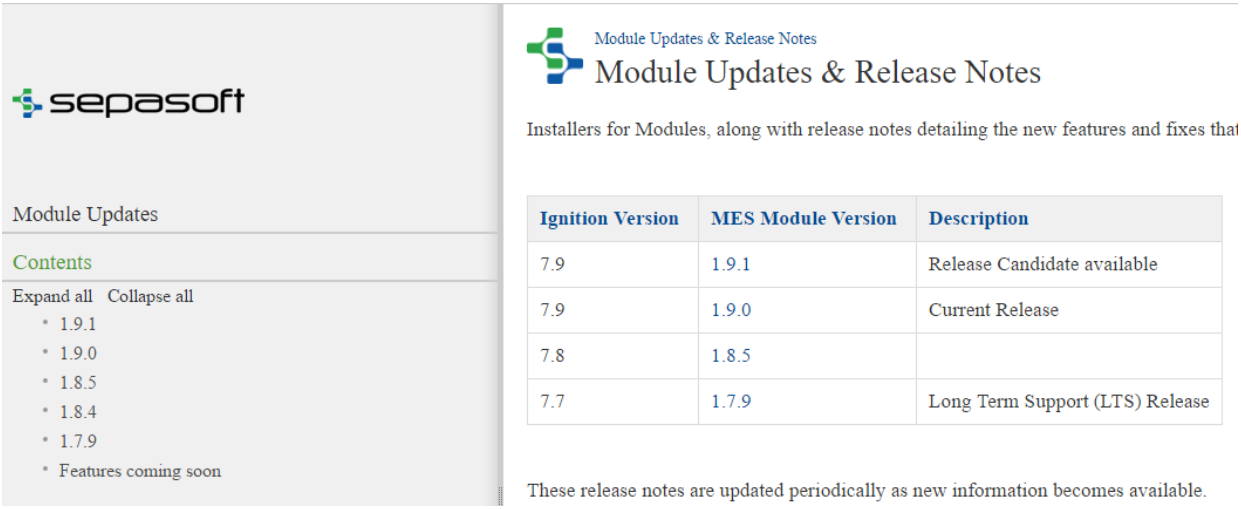
We strongly recommend keeping the MES data in a separate database from any other data that you may store as part of custom ignition applications. Keeping them separate ensures that table schemas are not modified that may break the MES Module functionality. It also provides the ability to troubleshoot database connection issues through the Ignition Gateway by identifying the source of mal-formed queries that may be affecting Gateway performance.

> **Info**
>
> The recommended procedure to change the database on an existing system, is to stop current production in the MES system, disable the production model in the designer, change the data setting, re-enable the production model and restart production.

5.4 Installing the Production Simulator

If you don't have a real PLC to work with, the MES Modules come with a simulator that allows us to simulate any kind of data. The production simulator doesn't get installed from the Ignition installer. We have to download and install the module separately.

1. First let's download the module from Sepasoft downloads page. Under the modules section, you will find a section at the bottom called **MES Modules** where you can pick which the module called **Production Simulator-module.modl.**

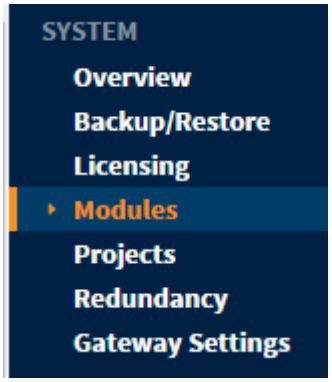


Here's a video on how to install a module

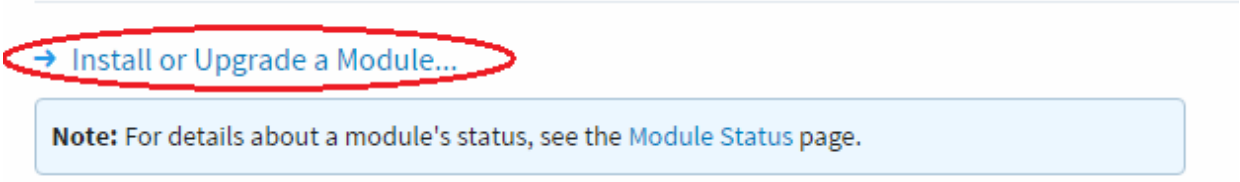
Watch the Video

Module Installation

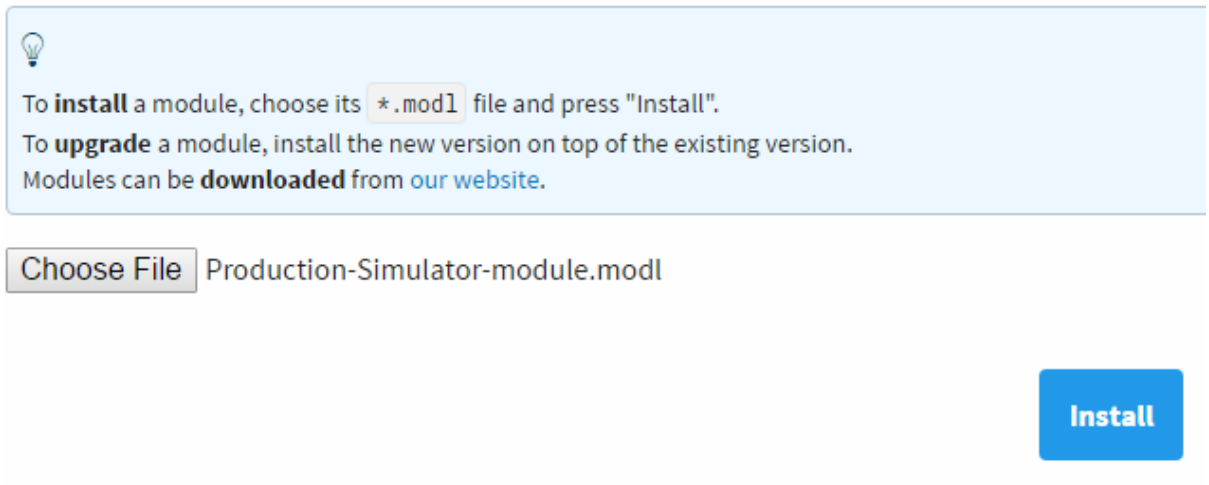
2. Once you have the module downloaded, log in to the Ignition Gateway webpage and select **Configuration > Modules** from the menu. There you can install modules (.modl files) one at a time.



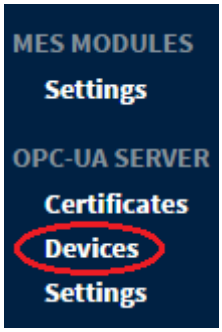
3. Scroll down to the bottom of the list and click on **Install or Upgrade a Module....**



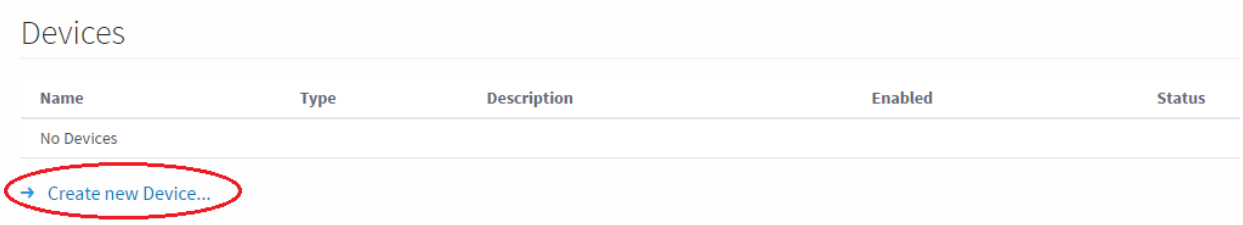
4. Browse for the simulator module and click Install. Now that the module is installed, we have to add a device to Ignition that acts like a PLC.



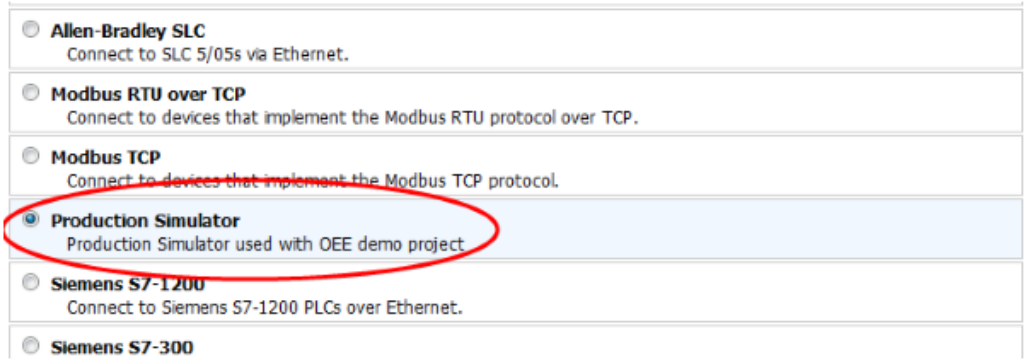
5. In the configuration area select **OPC-UA > Devices** from the menu.



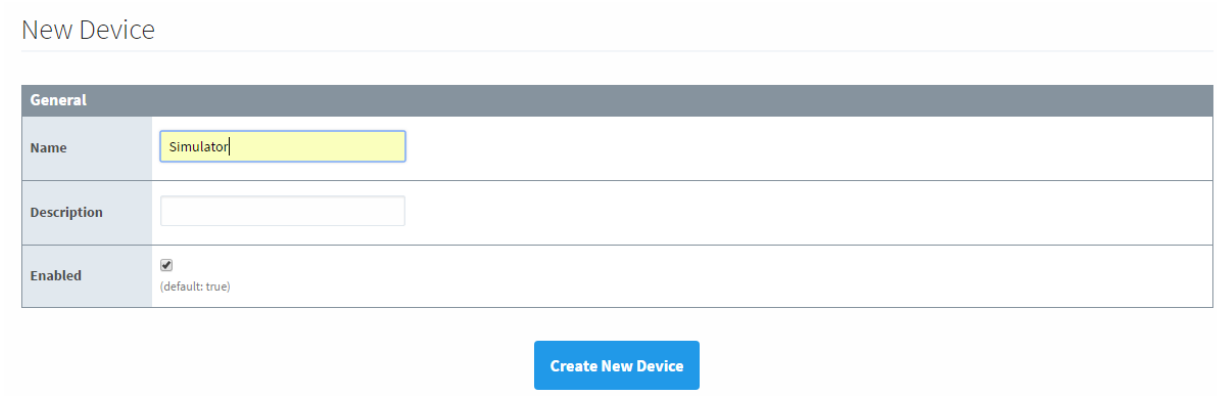
6. Click on the **Create new Device...** link.



7. Select the **Production Simulator** driver.



8. Name it **Simulator** and create the device.



5.4.1 Installing Simulation Files

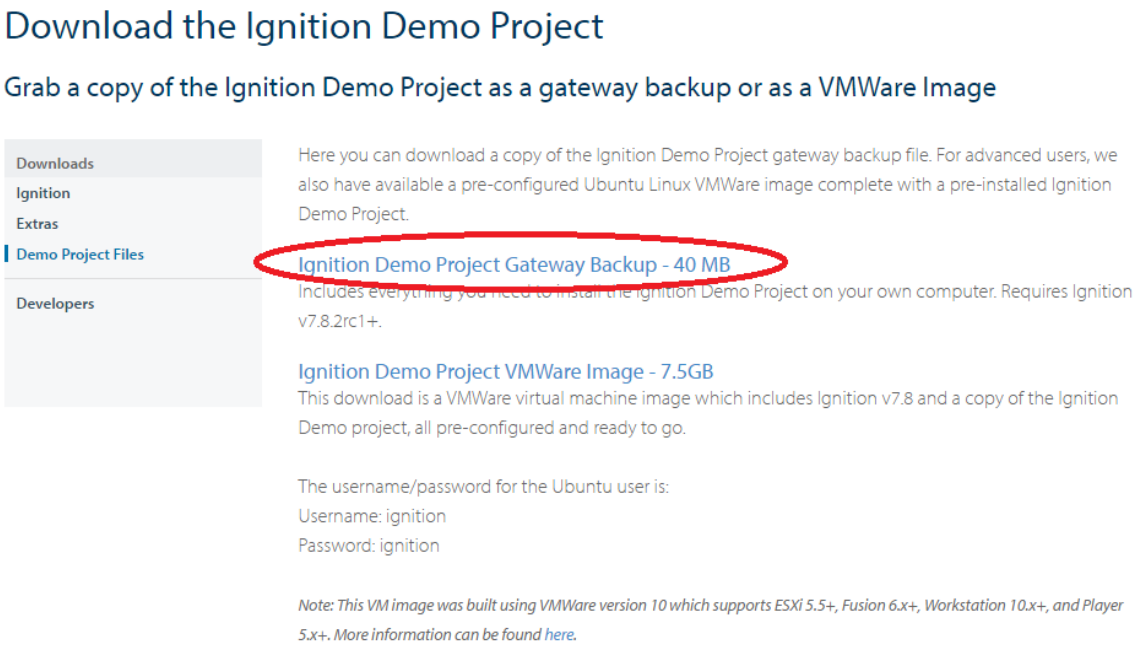
The production simulator is completely driven off of CSV files. Inductive Automation provides a few CSV files as examples to help you get started. First let's download the samples from

https://inductiveautomation.com/downloads/demo-project-files

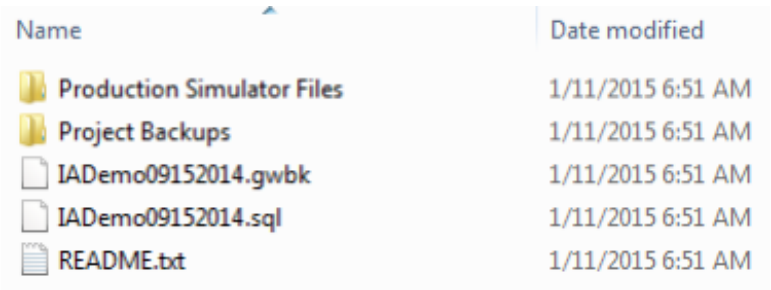
Click on the **Ignition Demo Project Gateway Backup** to download the demo projects which include the simulation CSV files.

Download the Ignition Demo Project

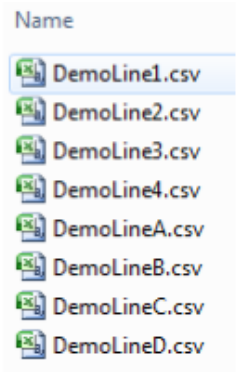
Grab a copy of the Ignition Demo Project as a gateway backup or as a VMWare Image.



Once the ZIP file is downloaded, unzip it to any location. The contents should look like the following:



You will find 8 CSV files inside of the **Production Simulator Files** folder.



Copy all 8 CSV files to: **C:\Program Files\Inductive Automation\Ignition\data\drivers**. Create a **drivers** directory is one doesn't already exist. Once the files are copied click on the edit button to the right of the device to reload the CSV files. Simply press **Save Changes** to reload the simulator.

OPC Quick Client

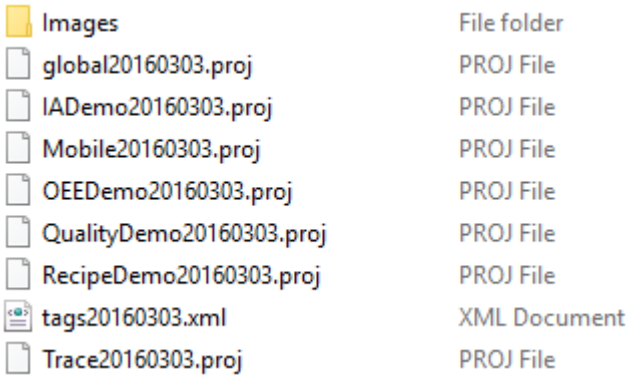
To verify the simulator is working correctly click on the **OPC Connections > Quick Client** in the configuration area. Locate the simulator device by expanding the tree starting from **Ignition OPC-UA Server**. You should see a separate folder for all 8 CSV files.

5.5 Installing the MES Sample Projects

Sepasoft provides a sample project for each MES Module. The sample project is a perfect starting point. Rather than starting from scratch, you can start with the sample project and customize it to fit your needs.

To install sample projects, go to the inductive automation website. Click on downloads. Then click Demo project files. The first link you see would be the Ignition Demo Project Gateway Backup, just click on it. Download the zip file and unzip it. Now you can see your files in the documents library, inside the folder called project backups.

The contents should look like the following:

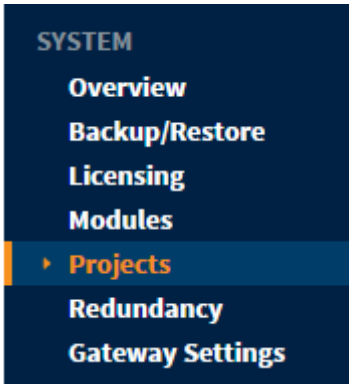


Here's a video to install the sample project.

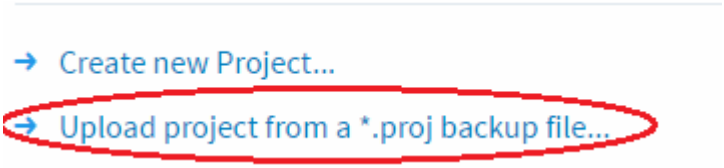
Watch the Video

Sample Project Installation

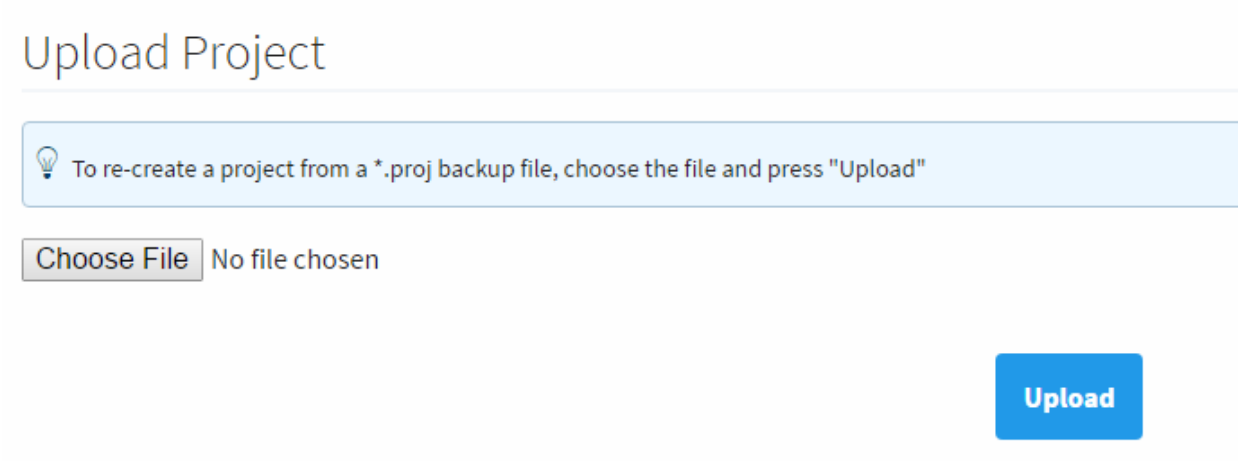
There is a separate project backup (.proj file) for each MES Module. Now to install each sample project, login to the Ignition gateway configuration page. In there on left side configuration, go to Projects.



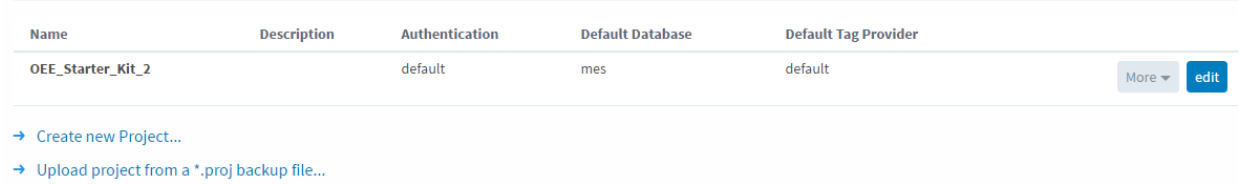
There you can see all the projects, Click on the **Upload project from a *.proj backup file...** link, choose the *.proj backup file.



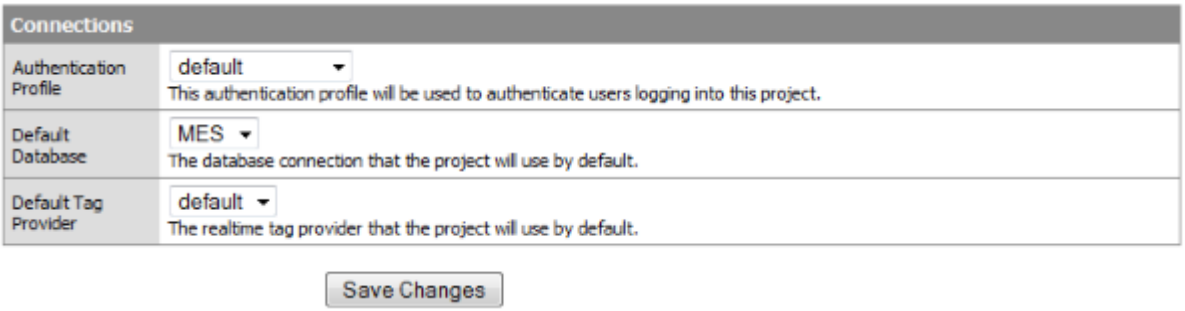
Then click on the Upload button and make sure that the authentication and the default database are corrected, if there are no match on the server.



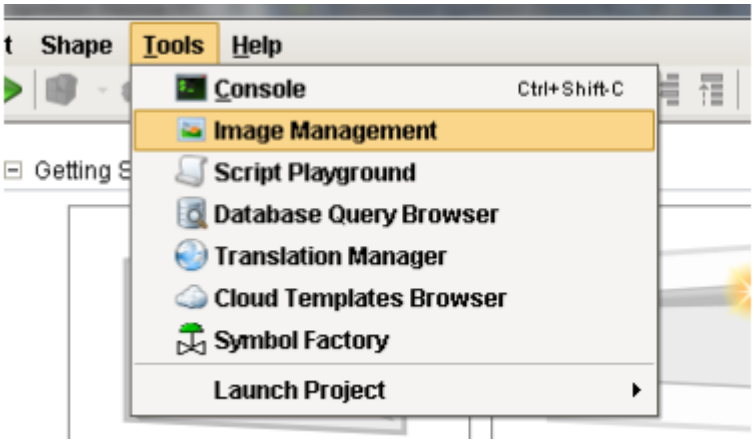
Press the edit symbol if you have a warning.



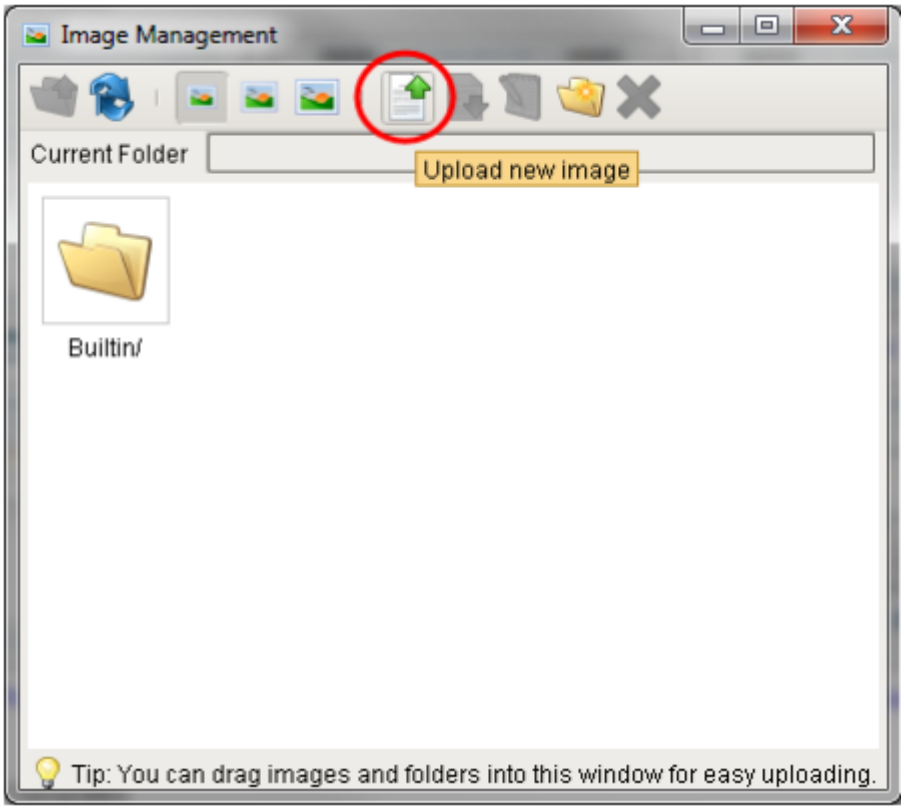
Choose the **default** authentication profile and press save. Now when you log into Ignition designer, you can see the screens provided by the sample project.



Once all of the projects are installed we need to open the Ignition Designer to upload the images since the project uses icons that don't come with Ignition. Launch the Ignition Designer and select any project to edit. In the designer choose **Tools > Image Management** from the top menu.

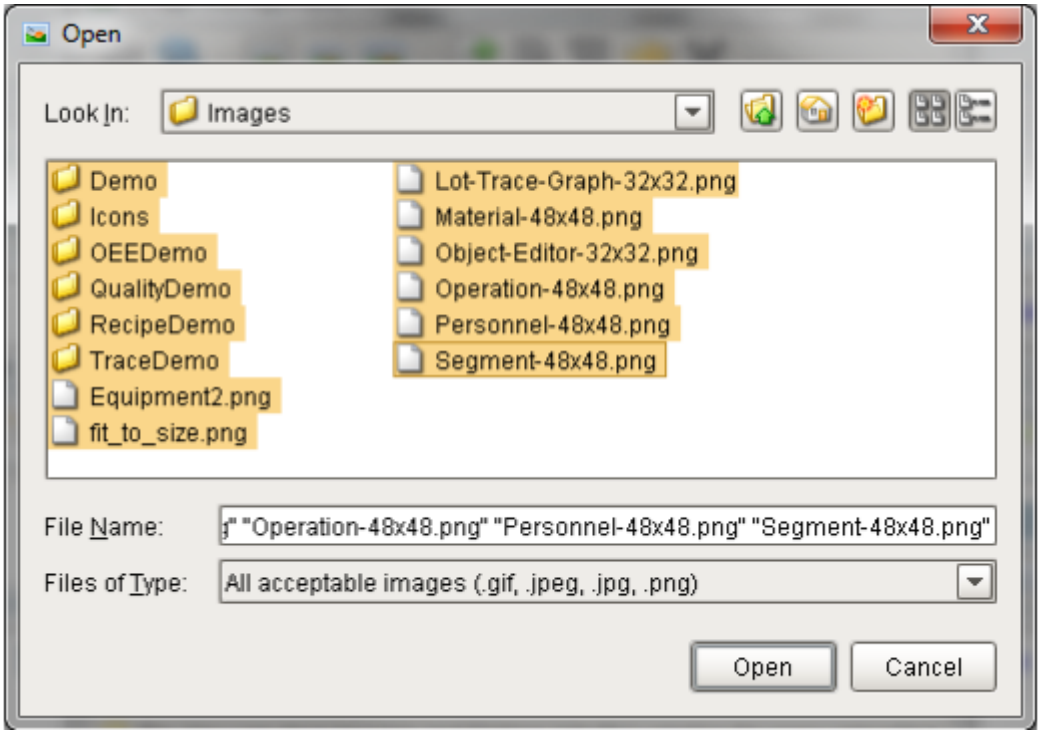


Press the **Upload new image** button.



Navigate to the images folder inside of the project backup you downloaded. Select every folder and image inside of the images folder and press **Open**.

Now all of the images we need are loaded.



5.6 Licensing and Activation

5.6.1 Trial Mode

The MES modules follow the same trial operation as Ignition. Any MES module can be used for 2 hours at a time, with no restrictions. At the end of the trial period, the system will stop logging data to the database, display expired trial overlays on live values, and clients will see a demo screen. By logging into the gateway, you may re-start the demo period, and enable another 2 hours of execution. The demo period may be restarted any number of times.

You may install an unlicensed MES module into a licensed Ignition server. The Ignition server licensing will not be affected and the MES module will operate in Demo mode.

5.6.2 Licensing

The MES license can be purchased along with, or separately from, the Ignition license. Despite the modular licensing, each Ignition server only has a single CD-Key and license file. That is, there is a single license file that dictates which modules are current activated.

When module(s) are purchased, you will receive a CD-Key - a six digit code that identifies your purchase. You then use this CD-Key to activate the software through the Ignition Gateway. Activation is a process by which the CD-Key and its associated parameters get locked to the machine that you are activating. If you adding an additional module, your account will be updated, and you can re-use your existing CD-Key to activate the new features. For this reason, if you purchased the MES module separately from the Ignition server, the MES license will have to be added to your existing CD-Key.

It is possible to inactivate your CD-Key, freeing it for activation on a different machine.

5.6.3 OEE Module Licensing

This powerful module combines overall equipment effectiveness (OEE) calculations and downtime tracking to help operations managers measure efficiency and gain insight on driving continuous improvement activities.

• Downtime Data Collection

• Real-Time Efficiency Tracking

• Monitor Asset Utilization

Not all production facilities have the large number of lines and cells while others do. For this reason there are two basic editions to choose from to meet your situation:

License	Description
Machine	One active OEE calculation per machine license with downtime collection for an unlimited number of child cells (sub-machines)
Site	Unlimited OEE calculations per physical production site and Ignition server license
Enterprise	Connect multiple MES Ignition Gateways across your entire enterprise to form a large, centrally managed MES solution. Analyze MES data from multiple production facilities at the enterprise server. One license per Ignition gateway server is required

5.6.4 Track & Trace Licensing

This paperless, fully integrated solution can provide production control and track product from the raw materials to the finished state, access genealogy data, and set up a centralized operator interface for all MES information.

• Data Connections & Visual Trace Graph

• In-Depth Data and Analysis

• Aligns with ISA-95

License	Description
Machine	One active operation (task) per machine license with unlimited segments (sub-tasks) under the operation
Site	Unlimited active operations per physical production site and Ignition server license

5.6.5 SPC Licensing

Ensure that statistical process control (SPC) data is accurately collected on time, every time by using the powerful features of the SPC Module. Deliver real-time SPC data in a comprehensive format using the flexible control charts and analysis tools.

• Automatic Sample Scheduling

• Automatic Rule Evaluation

• Powerful SPC Control Charts

License	Description
Machine	Sample collection, SPC rule monitoring, sample scheduling and viewing samples in control charts are limited to one machine per machine license. Includes basic control charts.
Site	Unlimited sample collection, SPC rule monitoring, sample scheduling and viewing samples in control charts per physical production site and Ignition server license. Includes basic and advanced SPC control charts (advanced control charts include process capability, process performance and box and whisker).

5.6.6 Recipe Management Licensing

Expertly build, manage and monitor your recipes. Easily manage product changeovers with a powerful recipe builder. Use multiple-level master recipes to instantly change several recipes simultaneously.

• Manage Recipes

• Audit Recipe Changes

• Real-Time Recipe Variances

License	Description
Machine	One active recipe per machine license
Site	Unlimited active recipes per physical production site and Ignition server license

5.6.7 Activation

Activation, as mentioned above, is the method by which a cd-key is locked down to the install machine, and the modules are notified of their license state. It is a two step process that can be performed automatically over the internet, or manually through email or the Inductive Automation website.

Step 1 - Enter CD-Key

When the software is purchased, you are provided with a six digit CD-key. After logging into the gateway configuration, go to Licensing > Purchase or Activate, and select "Activate".

Enter your CD-key.

Step 2a - Activate over Internet

If your computer has internet access, activating over the internet is the easiest option. A secure file will be created with your cd-key, and sent to our servers. The response file will then be downloaded and installed, completing the entire process in seconds.

OR

Step 2b - Activate Manually

If you do not have internet access on the installation machine, you must activate manually. In this process, an activation request file is generated (activation_request.txt). You must then take this file to a machine with internet access, and email it to support@inductiveautomation.com, or visit our website to activate there. Either way will result in a license file (license.ipl) being generated, which you then must take back to the Gateway machine and enter into the License and Activation page.

6 MES Product Suite Overview

The Sepasoft MES Product Suite provides a modular approach to building your own Enterprise Class MES solution. We provide you with OEE/Downtime Tracking, SPC, Production Scheduling, Recipe Management and Track & Trace modules that can function and operate by themselves, or integrate seamlessly when installed together. The modules provide everything needed to build and deliver a stand-alone MES application, or can be integrated quite nicely with third party ERP, Inventory Management and Asset Management systems. The MES modules form an additional layer on the Ignition software stack, extending the standard functionality and also allowing for a rich and powerful way to customize the MES implementation to deliver virtually anything required for your MES solution.

6.1 What Is Ignition?

Ignition is a web-based software platform that is great for creating HMIs, SCADA, and MES applications. Ignition, is affordable and easy to get started with, while flexible and capable of scaling up to the largest projects. Ignition is installed as server software and is:

- **Web-based** Ignition is installed and deployed using web technologies.
- **Web-managed** Ignition's platform is managed through web pages.
- **Web-launched Designer and Clients** Ignition's Designer tool and network clients are launched using web.

For more information on Ignition, please go to www.inductiveautomation.com

6.2 Benefits of Ignition / MES Architecture

We don't provide an out-of-the-box MES solution. Some assembly is required, and that's ok because every manufacturing operation will have something unique about it that you simply can't account for or handle through a configurable solution. Whether it is the process data to be tracked, materials used, data sources, which ERP system or PLC you need to connect to, or how KPI metrics are calculated, every manufacturer needs something special and specific to their process and the way they run their business. Even within the same organization, different plants and different departments needs a level of customization 'outside of the box'.

The Sepasoft MES Solution will not provide everything you need, but it will get you close to your goal and very quickly there. What is not provided can then be easily implemented using the standard ignition components and scripting functions. Whatever your need, you can be confident that the Sepasoft MES / Ignition solution will allow you to create the MES solution that your business needs.

6.3 Ignition Key Points

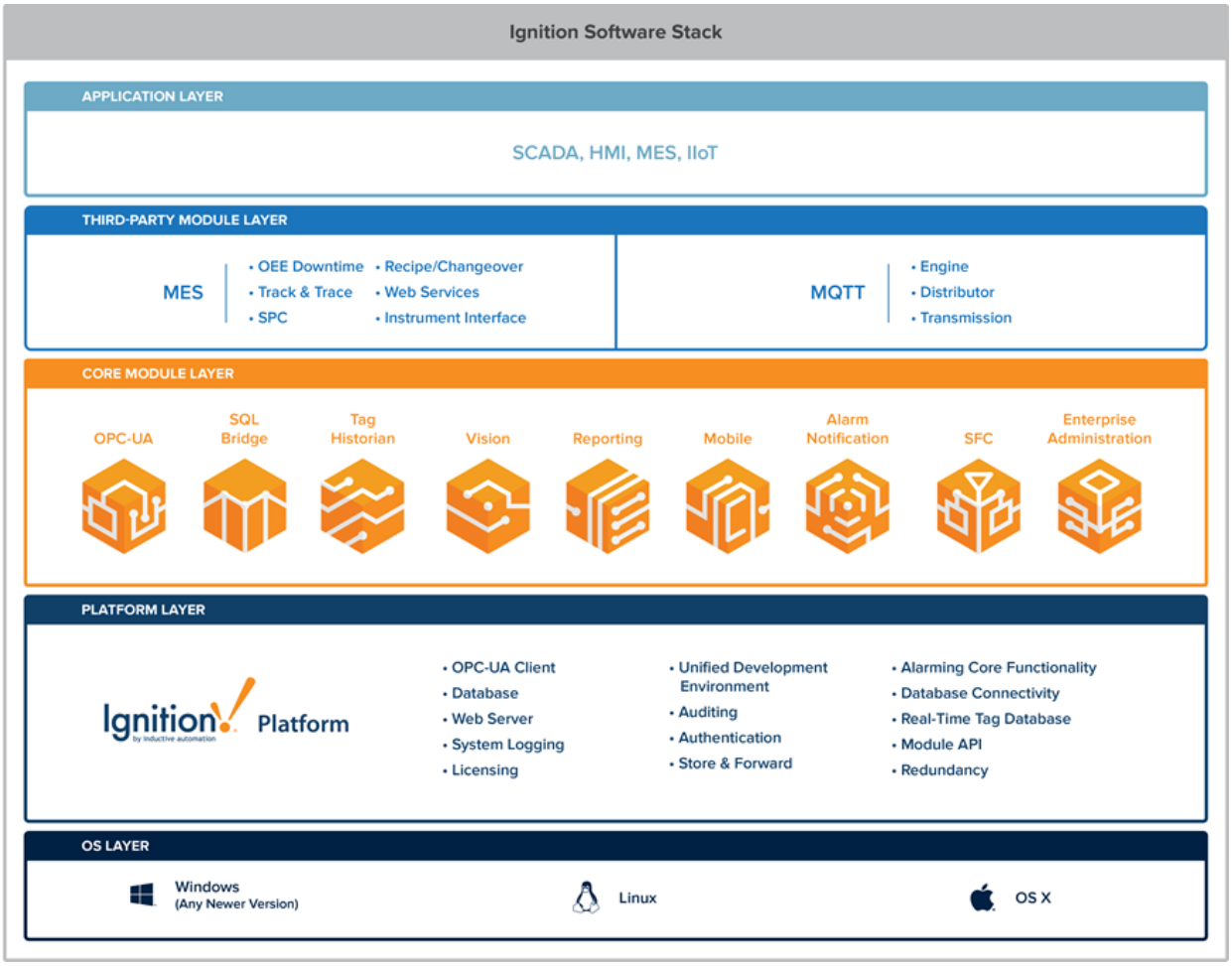
- **IT Friendly**
  - Client-Server Architecture
  - Scalable
  - AD Authentication & Security
  - Database Driven & Database Agnostic
  - Leverage your internal IT resources
- **Rapid Deployment**
  - Web-Based Application
  - Web-Based Designer
- **Business Model**
  - Unlimited Clients & Tags
- **Feature Rich**
  - Process Historian
  - Alarm Notification & SMS messaging
  - Reporting
  - Mobile

6.4 Sepasoft Key Points

- **Modular**
  - Build your MES project in phases
- **Customizable**
  - Build exactly what your client needs
- **Connectivity**

- Connect Enterprise Information Systems and Manufacturing Equipment together

- **ISA – 95 Compliant**



6.5 What is MES?

6.5.1 Overview

Manufacturing Execution Systems or MES software are tools designed to help a company manage its manufacturing processes. Manufacturing is too complex nowadays to not have a system in place that defines how customer orders are scheduled and manufactured on the production floor. If you're in manufacturing, then you have MES in place, it may just be your implementation is in the form of spreadsheets, emails and posted schedules. Modern MES software helps by replacing those spreadsheets with a connected system that provides real-time operations information to those who need it when they need it, and allows the flow of data from plant floor devices and equipment directly up to ERP and Inventory Management Systems.

MES system is not the system of a plant floor and it is not an ERP system. Manufacturing execution systems or manufacturing operations management (MOM) software is modeled after the ISA-95 standard and is designed to bridge the communication between the plant floor and management in executive levels. Most companies have a little of any success in standardizing all the data in MES layer. Ask yourself, how is your company handling MES data today. Do you have separate SPC systems, track and trace on paper,settings managed at local operation interfaces, spreadsheets using schedules? The data coming from different sources can create confusion and system which doesn't work together cause a lot of frustration. The result is a loss of money, time and quality. So what is missing here? The answer is a fully integrated unified MES solution where you can get all your MES data in one place, on one screen. Now with Sepasoft MES you could finally do just that.

6.5.2 What makes our MES suite of products so special?

- Real-Time Efficiency Tracking and Monitoring Controls
- Downtime Data Collection
- At-a-Glance Executive Dashboards
- Easy ERP Integration
- Mobile MES Access
- Production Scheduling
- Quality and Maintenance Management
- Batch Processing
- Fast, Customizable Implementation
- Automatic Data Collection

MES operates across multiple function areas, for example, management of product definitions across the product life-cycle, resource scheduling, order execution and dispatch, production analysis and downtime management for Overall Equipment Effectiveness(OEE), Product Quality, Track and Trace.

MES creates the 'as-built' record, capturing the data, processes and outcomes of the manufacturing process and maintaining the system of record. This can be especially important in regulated industries, such as food and beverage or pharmaceutical, where the documentation and proof of processes, events and actions may be required.

The idea of MES might be seen as an intermediate step between, an Enterprise Resource Planning (ERP) system, and a Supervisory Control and Data Acquisition (SCADA) or process control system, although historically, exact boundaries have fluctuated. Our MES solution is targeted at the Plant Operations layer in manufacturing systems, providing flexible methods of interacting with the ERP system and the plant equipment. According to ISA-95 models, plant operations comprise Production Operations, Inventory Operations, Quality Operations and Maintenance (or Engineering) Operations. In each of these areas, our MES Product suite has modules that provide the necessary functionality.

MES is the factory floor execution system. It is the layer in which the operators directly interact to step through the execution of the work flow to produce or repair product. The list of work to be performed, specific instructions to execute the work, data points to be collected, quality inspections of the work, sign off’s indicating the work is complete, are all performed within this layer.

MES provides the workflow, visibility and event notification required to ensure that manufacturing is meeting enterprise information demands. Simultaneously MES reduces non value-add activities, increases data accuracy and provides the ERP system with real-time data needed to maximize enterprise processing, planning and scheduling activities.

MES system acts as a messenger between the factory floor, corporate engineering (PLM), and corporate planners / schedulers (ERP). When operator requires data from the ERP or PLM: initiates the request within MES; the MES system then connects to the appropriate system to retrieve and display the information.

With Sepasoft MES, now you can have a system that is capable of being friendly with your ERP system sharing production schedule with all your departments, reviewing numbers, info and various types of data in a unified system without any need to compare paper reports. The connected docs would help you to relate the data that is used during the manufacturing processes. All modules integrate seamlessly with the Ignition HMI and SCADA softwares. You can achieve true collaboration and access all your data from one unified system.

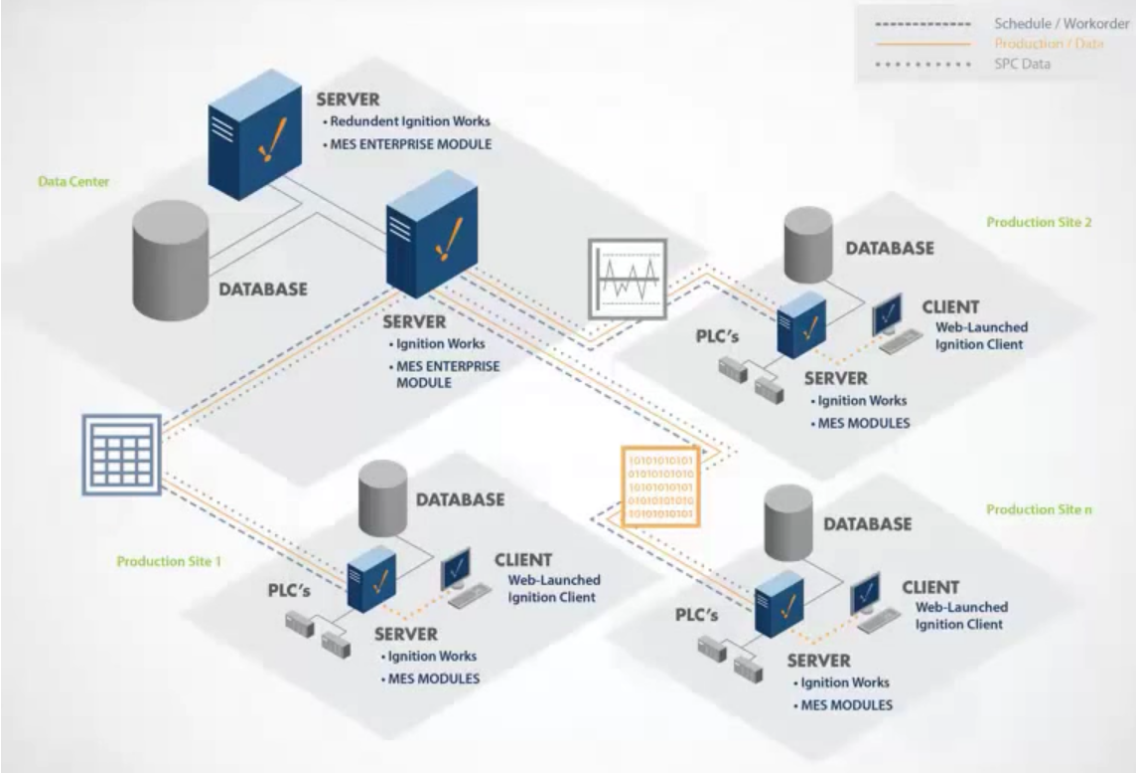
6.6 Understanding MES Architectures

There are many possible system architectures as to how MES modules are installed and how they interact with other key systems. Most users begin using a standard architecture, where MES and all components are all installed on a single machine. As your system expands, you can investigate into other possible architectures .

6.6.1 Enterprise Architecture

If you have multiple production sites to meet MES functionality and if you also want to collect data in a central location such as the headquarters, you must go with an enterprise architecture.

In the architecture shown below, you can see a data center and three production sites. The production data from each sites is to be stored in the data center. The production sites must run independently from each other and have to run even when the communication with the center site goes down. So in this situation it is really important that all the production sites have an Ignition server by themselves.



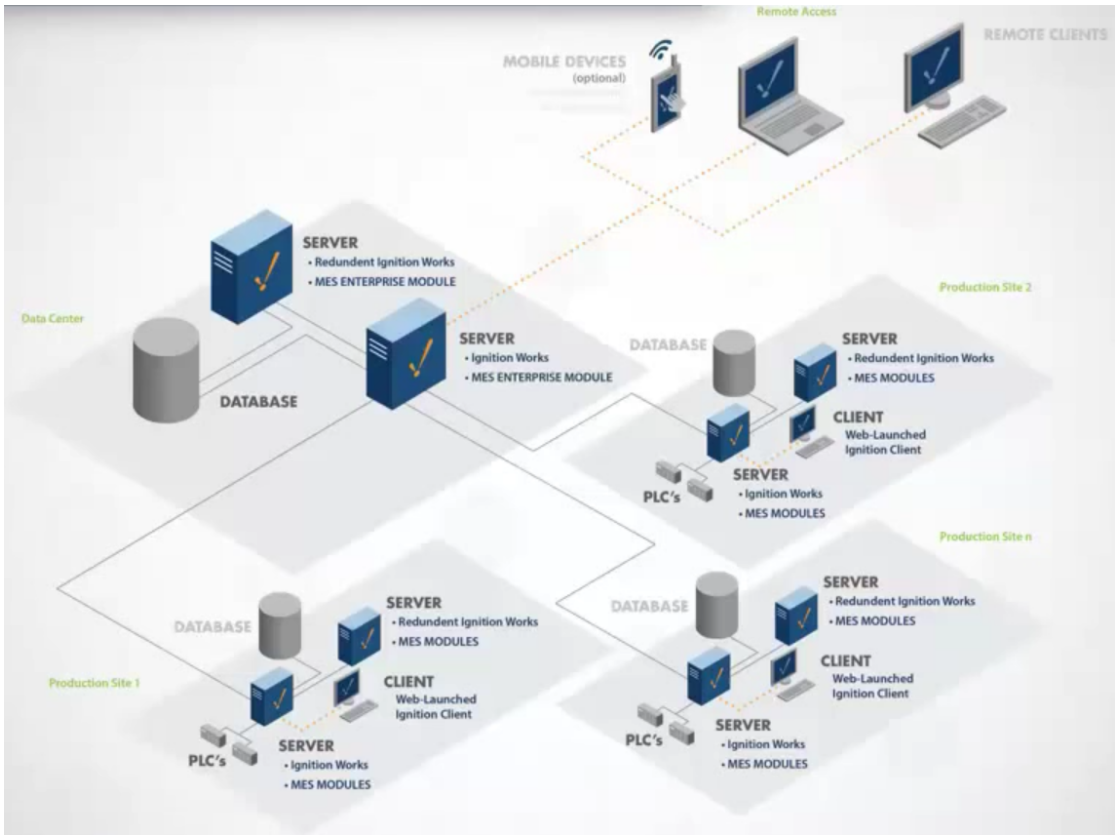
In this Page

A lot of people get annoyed with having a single set of servers at the data centers for all the production sites. The problem with this is that if a site loses communication to the central server, it's going to be blank and no longer it can collect the MES data. So we should have servers at each local production sites.

Now we have a local database for each production site. How can we get the MES data in the local data center reflect in the global data center also? That's where a setting in the MES modules comes into play. There is a way to perform all the data analysis locally as well as remote. All three production sites storing the information locally, push the data up to the data center. If anyone of them lose connection to the data center then we can use the cache it had on the local database to have a backup.

6.6.2 Redundant Architecture

The redundant architecture will have a couple of Ignition servers in the data center. All the production sites will have an additional server as well. This will help in the secure data storage. If one server goes down, the other will take up the charge. In other words, if the primary fails, it's backup will take over and will continue the run. It is very efficient since it prevent data loss. It assures a fault tolerant system for MES.

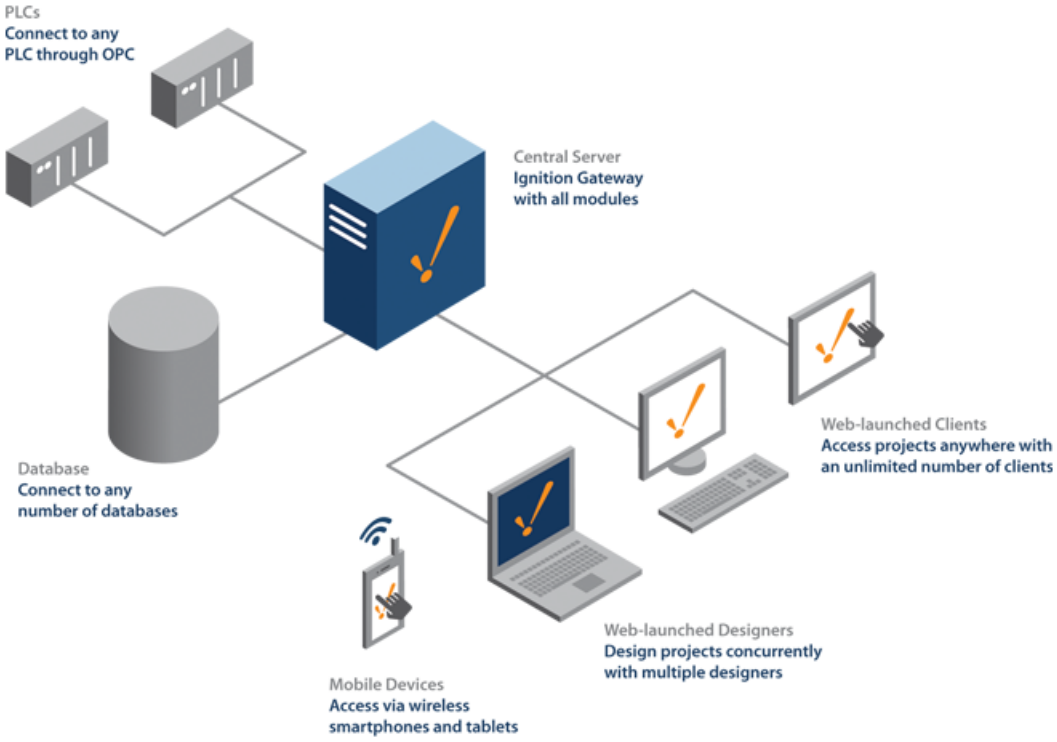


How you can do redundancy on Ignition servers?

Wherever you have single Ignition server, add one and simply make them redundant. It is very easy to set them up. You just have to enable it and turn on the setting, Ignition will synchronize the projects for you. The Ignition platform provides redundancy, which means that all the MES modules automatically get redundancy built into them.

6.6.3 Standard Architecture

The simplest architecture for MES is to have a single Ignition server at a single site. It is perfect for companies to have a single site in order to collect MES data. In the figure below, we have a single Ignition server with all the HMI, SCADA and MES modules installed. From the server you can connect to one or more PLCs to collect the MES data within that site. You can also connect to one or more SQL databases where the MES data is to be stored. You can store both the run time data and the analysis data in the same databases or separate databases.



All the configurations and the screens for each clients are going to be in a single server. So within this site you can open up any number of runtimes inorder to view the MES data, to start production run and to collect samples. You can also open up a client at home through VPN connection or do a connection to the local LAN there. It's a very simple architecture and its common for companies who have independent sites where they wanted to work on themselves.

6.7 ISA-95 Overview

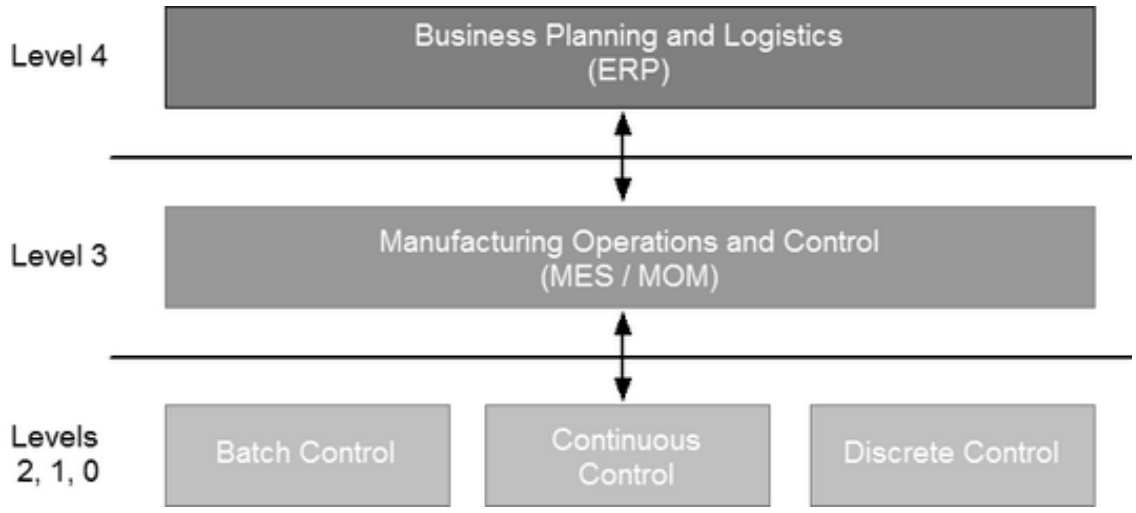
The ISA-95 standard describes the interface content between manufacturing operations and control functions and other enterprise functions. The goals are to increase uniformity and consistency of interface terminology and reduce the risk, cost, and errors associated with

implementing these interfaces. The standard can be used to reduce the effort associated with implementing new product offerings. It is an excellent starting point for successfully implementing interfaces between enterprise (ERP) and control systems (SCADA) through the MES Layer.

ISA originally stood for **Instrument Society of America** and as a group, they have set many standards used for automation. Today, ISA have evolved into more than just instruments and beyond America, and as a result changed their acronym to stand for **International Society of Automation**.

The Sepasoft OEE 2.0 and Track and Trace Modules are specifically built around the ISA-95 standard that was developed to automate the interface between enterprise financial systems and control systems on the plant floor. Information that the top level system or ERP (Enterprise Resource Planning) has about upcoming production requirements is needed on the plant floor. Likewise, some of the production details from the plant floor is valuable at the ERP level.

Plant floor control systems are designed to control processes and machines and are not well suited to handle much production data. They can make control decisions in the 5ms to 200ms range, but have limited historical storage and database capabilities. ERP systems tend to be more transactional based and are well suited to processed financial, inventory, receivables, etc. They do a great job of accepting orders, checking if additional raw material should be ordered, paying vendors, reporting and etc. that can be updated anytime during a day, week, month, quarter or even year.



Both the speed of the plant floor control systems and the planning and tracking ability of an ERP system is needed in the middle ground. This middle ground is commonly referred to as MES (Manufacturing Execution System) and MOM (Manufacturing Operations Management).

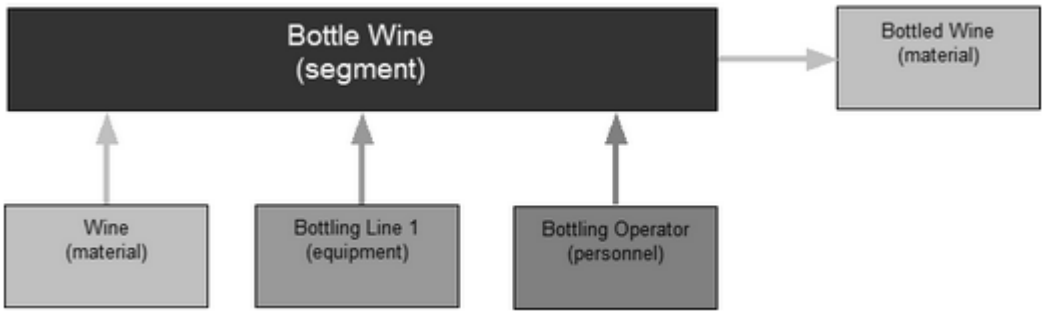
The objectives of the ISA-95 standard are to provide a consistent operational model and terminology that is a foundation for the different levels to communicate. In addition, systems on the same level can communicate in a consistent manner. It was developed to be applied in all industries and all sorts of batch processes, continuous processes and discrete manufacturing.

In this era of manufacturing, data stored in proprietary systems or in systems where it requires hours in custom programming to share data between systems, will not provide what is needed. In this era, manufacturing information systems must be data centric solution. Operations has their wish list and the cost, schedule and risk to create such a system is monumental and as a result typically doesn't happen. In other cases, funding is approved and the project is started, but it falls short of expectations. There has to be an easier way!

The Sepasoft OEE 2.0 and Track and Trace Modules are aligned with the ISA-95 model. If we started from scratch and developed our own, which was tempting because modeling after the ISA-95 standard is not an easy task, we would be yet another company with a different model that others have to learn and adapt to. If we did, we would be doing our customers and the industry a dis-service.

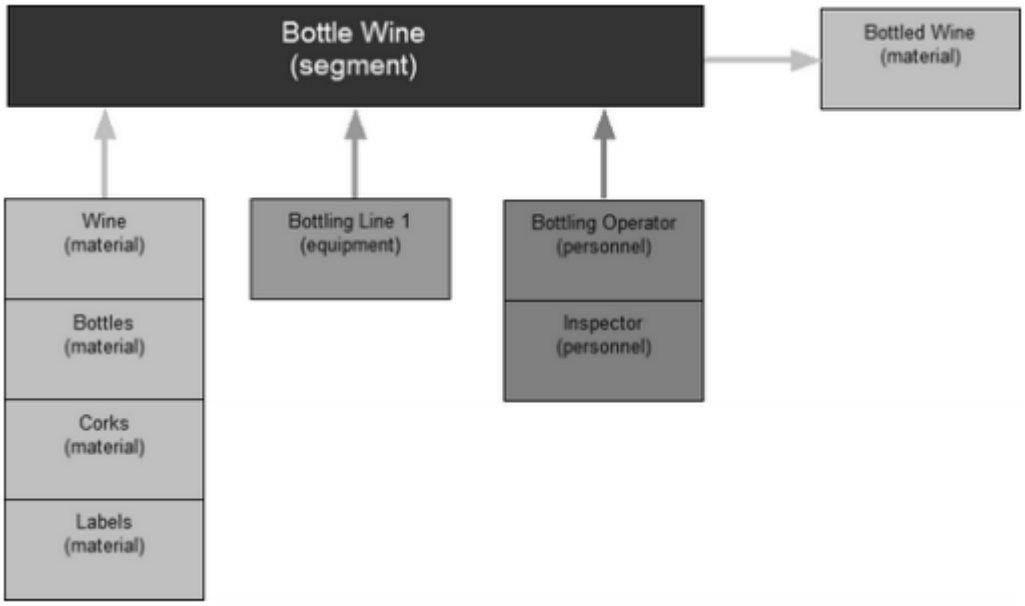
6.7.1 Basic Production Tasks

Any task that is done during manufacturing, requires resources. Resources can be material, equipment and personnel. The image below shows a basic segment of Bottling Wine and has three input resources of Wine (material), Bottling Line 1 (equipment) and Bottling Operator (personnel). Then there is one output material resource of Bottled Wine. Most often there is only one output material resource, but in some situations there can be multiple output resources. If not much trace detail is needed, then this is the minimum that is needed to run production. Actually, for the Track and Trace Module, material and equipment are required but the personnel is optional.



Basic Segment and Associated Resources

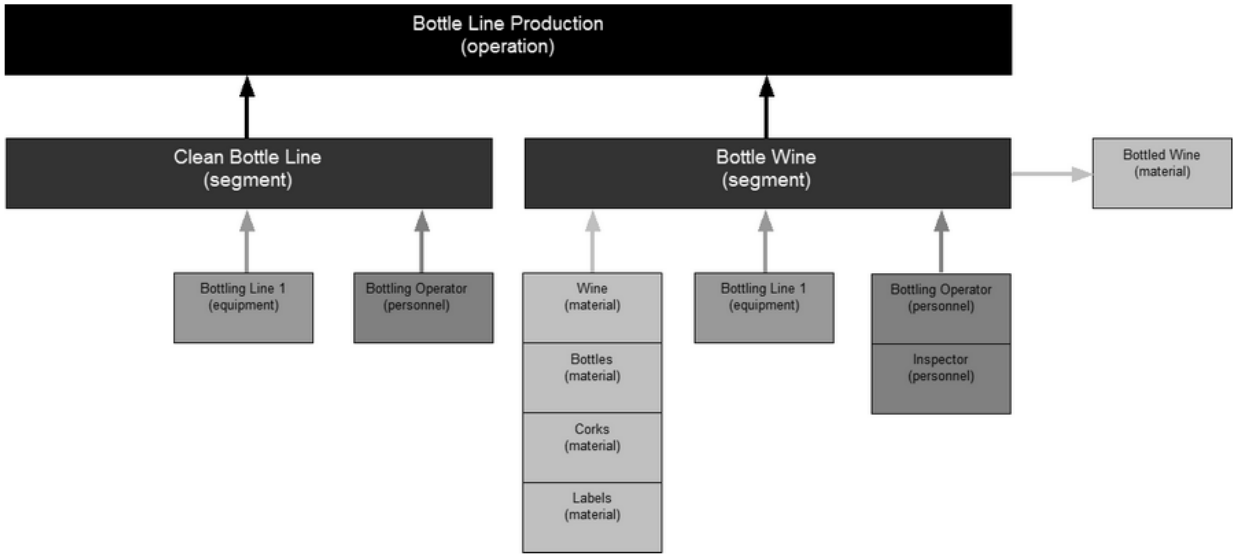
If more detail is desired, then more resources can be added to the Bottle Wine segment. Below we see Bottles, Corks and Labels, all of which are material, have been added to the segment. Also, another Inspector has been added as personnel. There is no limit to the number of resources that can be added to a segment. However, keep in mind that details about the resources have to be entered during production.



More Detailed Segment and Associated Resources

6.7.2 Advanced Production Tasks

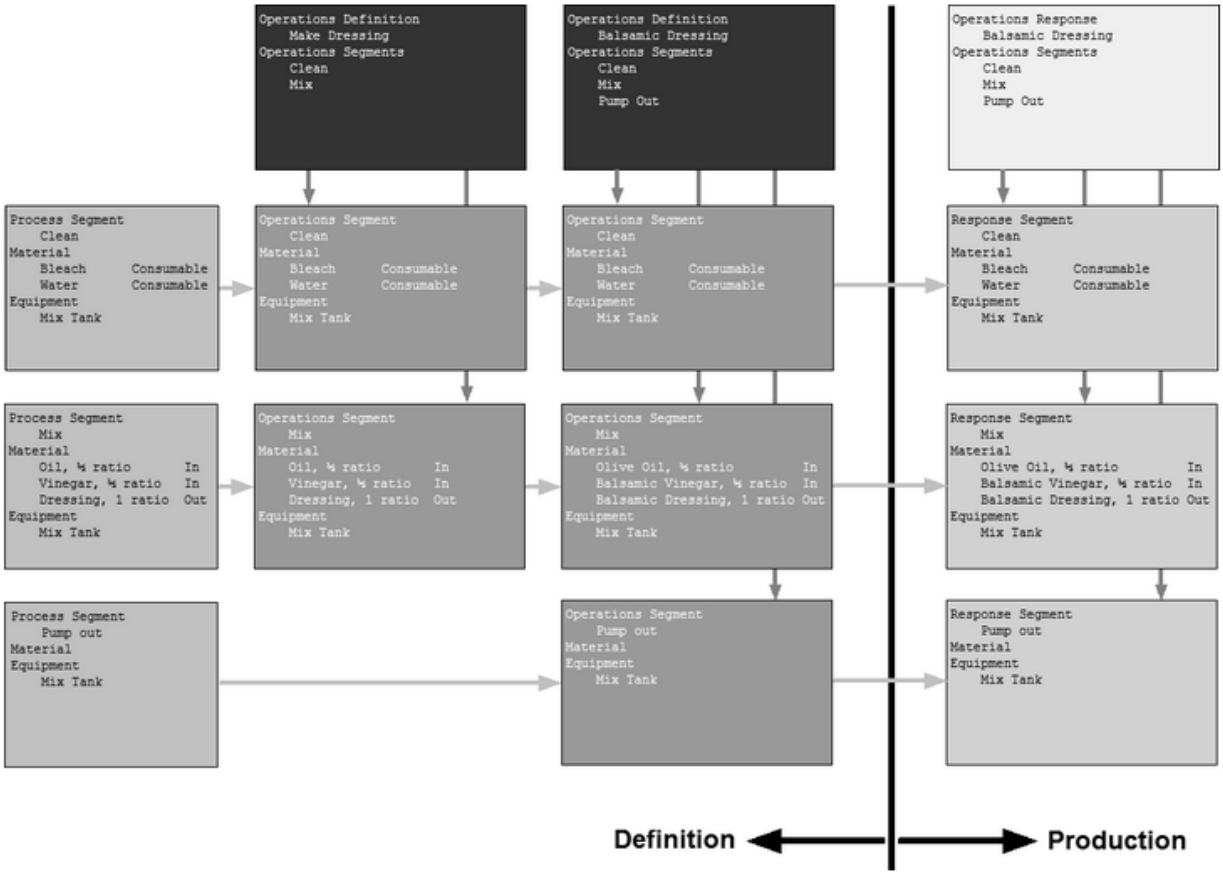
Production is not always as simple as run production, and might involve more that just one step. This is accomplished by including multiple segments into an operation. Remember that segments are the basic tasks. Now we can link multiple segments (tasks) together into an operation to perform more complicated tasks as shown in the image below.



Operation and Associated Segments and Resources

The image below shows how Process Segments are used by Operations Segments and how Operation Definitions refer to Operations Segments. Collectively, these are in the definition side. They are only created or modified when users are defining their production process.

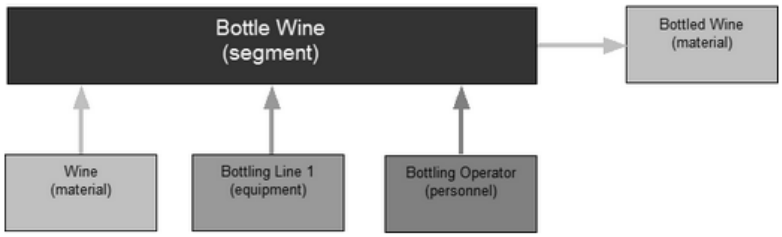
On the production side, Operations Response and Response Segments are created when the operator begins production. As a result, there will be a set of Operation Response and Response Segments for each production run.



Definition and Production

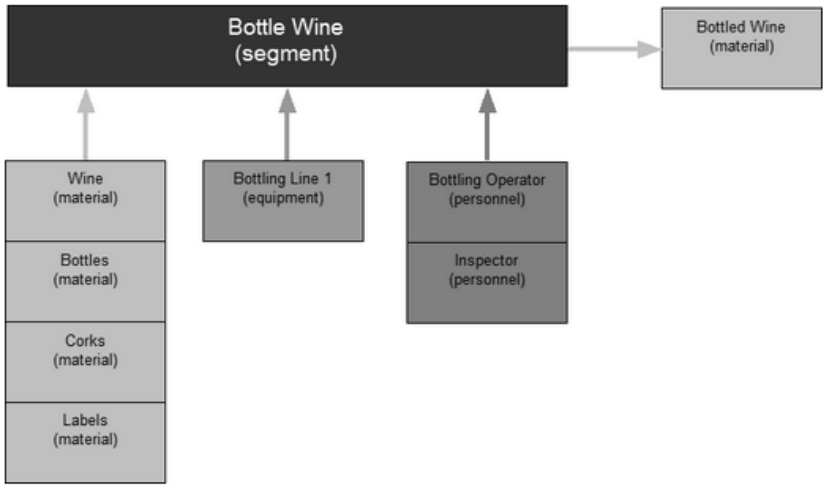
6.7.3 Resources

Any task that is done during manufacturing, requires resources. Resources can be material, equipment and personnel. The image below shows a basic task of Bottling Wine and has three resources of Wine (material), Bottling Line 1 (equipment) and Bottling Operator (personnel). It also shows the resource Bottled Wine (material) that was created from the task.



Basic Segment and Associated Resources

If more detail is desired, then more resources can be added to the Bottle Wine task. Below we see Bottles, Corks and Labels, all of which are material, have been added. Also, another Inspector has been added as personnel. There is no limit to the number of resources that can be added. However, keep in mind that details about the resources have to be entered during production impacting the production staff.



More Detailed Segment and Associated Resources

Equipment Resources

Equipment

Any automated production or processing that is done requires equipment. Manual production or processing is done at a location such as unloading at a dock. The dock is the location where the production or processing is taking place. It can also be manually adding a antenna at a work cell.

In the Sepasoft MES system, there are two types of equipment, processing equipment and tooling or rolling equipment.

Processing Equipment

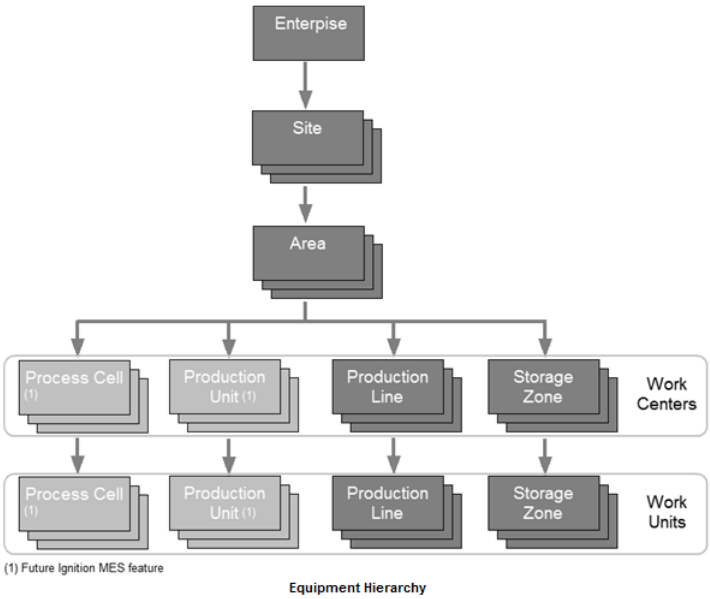
The processing equipment is defined in the production model and is defined using in the Ignition designer. The reason equipment is defined in the Ignition designer is because tags are used to collect production data, downtime, SPC sample data, etc. for it. Tags can only be assigned in the designer and also involves configuration in the control system, through the OPC server and tags to the equipment defined in the production model.

Supplemental Equipment

Supplement equipment, such as tooling and rolling equipment, can be defined either in the Ignition Designer or in the Ignition client. By allowing tooling or rolling equipment to be defined in the Ignition client, users are enabled to make dynamic changes without having to log into the Designer. This is handy for tooling such as dies, jigs, etc. but can also include rolling equipment such as forklifts.

Equipment Hierarchy

The processing equipment is organized into a hierarchy that starts at the top and works down to the equipment. The Sepasoft MES suite, uses this model to define equipment that is relatively permanent. This means equipment that tags are used to read information from and send control information down to during production. Because tags are involved, this type of equipment is defined in the Ignition Designer. Other rolling or tooling equipment that do not use tags can either be configured in the Ignition Designer, MES object editor or from the built-in scripting language.



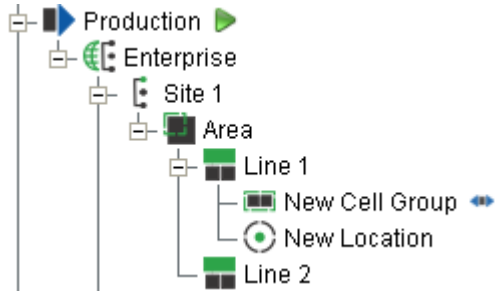
Equipment Hierarchy

The equipment defined in the production model is where production tasks can be done. The rolling or tooling equipment can be added as additional equipment resources use during the production tasks. For example, a press (Production Line defined in the production model) will have associated dies used for making various products. If tracking of which die was used for a production task, then the die can be defined as a rolling or tooling equipment item separate from the the press.

Production Model

In the Sepasoft MES system, the equipment hierarchy is defined in the production model. It is important to define what the production model is, because everything done in the Sepasoft MES system revolves around it.

A production model defines your manufacturing or process in tree view form. It provides an organized way to easily configure, control and analyze your facility. It does not control the possible routes that product flows within a facility, but simply is a organized manner in which to manage machinery.



Production Model Tree

MES Enterprise

The enterprise is the highest level of the production model and typically represents a manufacturing company. The Sepasoft MES system only supports one enterprise per Ignition server but can have one or more production facilities (sites).

MES Site

A site is a geographical production location and is part of an enterprise.

MES Area

An area is a physical or logical grouping of production types within a production facility.

MES Line

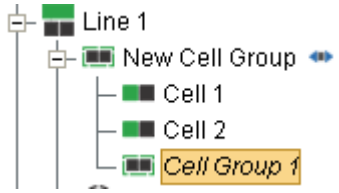
A line is a collection of one or more cells (machines) and/or cell groups (groups of machines) where product can be produced. It may have one or more machines or sub processes.

MES Line Cell

The cell is a single machine, sub process or step required in the manufacturing of a product.

MES Line Cell Group

A cell group contains two or more cells. Typically, these cells occur at the same time in the sequence of the line instead of one after another, causing the cell group to act as a single sub process or step within the production.



Cell Group Tree

MES Storage Zone

The storage zone is a space that material is stored. There can be multiple storage zones within an area. Each storage zone can have storage units where actual product is stored.

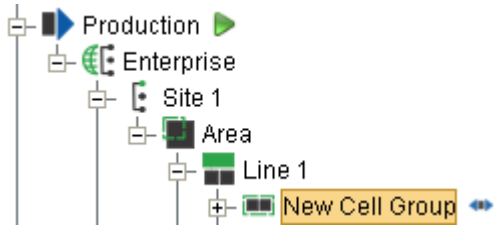
MES Storage Unit

The storage unit resides in a storage zone and provide greater granularity of where material is stored.

Equipment Path

In the Sepasoft MES system, specific equipment is normally referred to by the equipment path. It is simply the route starting with the enterprise and following down the items in the production model tree to the specific equipment item.

Referring to the production model in the image below, the equipment path to Line 1 is Your Enterprise/Your Site/Your Area/Line 1.



Equipment Path

Although it is not recommended, it is possible to have more than one line named Line 1. An equipment path will specify only one of the lines named Line 1, which is required for all Sepasoft MES functionality.

Equipment Classes

Defining production tasks for each specific piece of equipment, is very tedious. A better method would be to organize the equipment into categories, or class using ISA-95 terms. An example will make this clearer with fewer words. Consider five packaging lines in a packaging area and three of them can package mixed nuts and the remaining two cannot. Creating a mixed nuts equipment class with the three line within it, allows a single task to be defined specifying that a mixed nuts equipment resource is required.

Material Resources

Material

Any Production or processing that is done involves material. The material maybe raw material that goes into finished goods, or it can be consumable or by-product that is not directly related to the finished goods.

Material Definition

Material definitions are used to define raw materials, material that are partially processed but not in finished goods state and finished goods. Consider the following case: If we are assembling an electronic product, then we will have electronic components, including a circuit board, that will each have material definitions. The components will be soldered to the circuit board and will have a material definition for the sub assembly. Next, the circuit board will be added to the housing which will have a material definition that represents it. This will continue until the finished goods are complete. It may even include accessories that are sold with the finished product. Each will have a material definition. Think of this way, in order to know which lots of components were used to make a batch of circuit boards, then material definitions are needed.

Material Classes

Defining production tasks for each specific material, is very tedious. A better method would be to organize the material into categories, or class using ISA-95 terms. An example will make this clearer with fewer words. Consider unloading electronic components at a receiving dock. Defining a task to receive each type of component would be a management nightmare. Instead all of the components can be added to a Electronic Component class and when the operator does the receive components task at the dock, it prompts them for the specific component that belongs to the Electronic Components class. Only one receive components task has to be defined, which is much easier to manage.

Personnel Resources

Personnel

Any Production or processing that is done involves people. In the Sepasoft MES system, this is optional. If trace information about the personnel involved in the production or processing is desired, then it is supported. The person can be automatically selected based on their Ignition login or it can be selected by other means.

Person

The MES Person objects are automatically generated from the Ignition users that have first and last names defined. This prevents the default "admin" user from being created in the MES system and showing up in selection lists. When the Sepasoft MES modules first start, the MES Person objects are synchronized and then will be synchronized on a hourly basis thereafter. They can also synchronized on demand using a script function.

Person Classes

Defining production tasks for each specific person, is very tedious. A better method would be to organize the people into categories, or classes in ISA-95 terms. An example will make this clear with fewer words. Consider unloading vinegar at an unloading pump station. If there are ten operators who are qualified to unload vinegar, then creating a Vinegar Unload Operator class containing the ten qualified operators will require just one unload vinegar task definition. Adding an eleventh operator is as simple as adding that person to the Vinegar Unload Operator class.

6.8 Production Operations Management

ISA-95 defines production operations management as ...

'...the collection of activities that coordinate, direct, manage and track the functions that use raw materials, energy, equipment, personnel, and information to produce products, with the required costs, qualities, quantities, safety, and timeliness.'

Sepasoft's MES solution provides a modular approach to realizing ISA-95's goal for Production Operations Management. The following section looks at the Production Operations Management activity model and discusses which aspects are covered by the MES modules.

6.8.1 Product Definition Management

ISA-95 defines Product Definition Management as

'*...the collection of activities that manage all of the Level 3 information about the product required for manufacturing, including the product production rules*'. *Product Production Rules contain the information used to instruct a manufacturing operation on how to produce a product. This can include manufacturing or work instructions, recipes and product variant definitions.'*

The Sepasoft MES solution provides a number of modules that can be used separately or combined to handle all aspects of Product Definition Management.

Track & Trace Module

The Track & Trace module can be used to manage product definitions when material lot tracking (genealogy), WIP Inventory, production routing and production control of which materials can be consumed and created by manufacturing operations is required. Refer to the Track & Trace Overview for more details on this module.

OEE 2.0 Module

The OEE 2.0 module can also be used to manage product definitions when those materials will be used for operations for which OEE downtime tracking is required. Refer to Product Definition Configuration for more details on Product Definition Management with the OEE 2.0 module.

> ⚠ Both the Track & Trace and OEE 2.0 module use the same ISA-95 framework for the handling of product definitions. Materials created using the OEE 2.0 Material Manager are available for scheduled operations where OEE performance data is required and are also available for lot tracking and production control with the Track & Trace module. Materials created using the MES Object Editor for lot tracking are not available to OEE 2.0.

Recipe Module

The Recipe module allows for recipes to be created and associated with product definitions. Recipes can be created for product definitions containing machine settings that are downloaded to PLC's and provides variance monitoring as well as recipe management. Refer to Recipe Management Overview for more details.

6.8.2 Production Resource Management

ISA-95 defines Production Resource Management as '...*the collection of activities that manage the information about resources required production operations, including machines, tools, labor (with specific skill sets), materials and energy*'. The Sepasoft MES solution provides the Track & Trace module that can be used to build your Production Resource Management system. Please refer to Track & Trace Overview for more details.

6.8.3 Detailed Production Scheduling

ISA-95 defines Detailed Production Scheduling as

'...*the collection of activities that take the production schedule and determine the optimal use of local resources to meet the production schedule requirements*'.

The Sepasoft MES solution provides both the OEE 2.0 and Track & Trace modules that can be used to build your Detailed Production Scheduling system.

Track & Trace Module

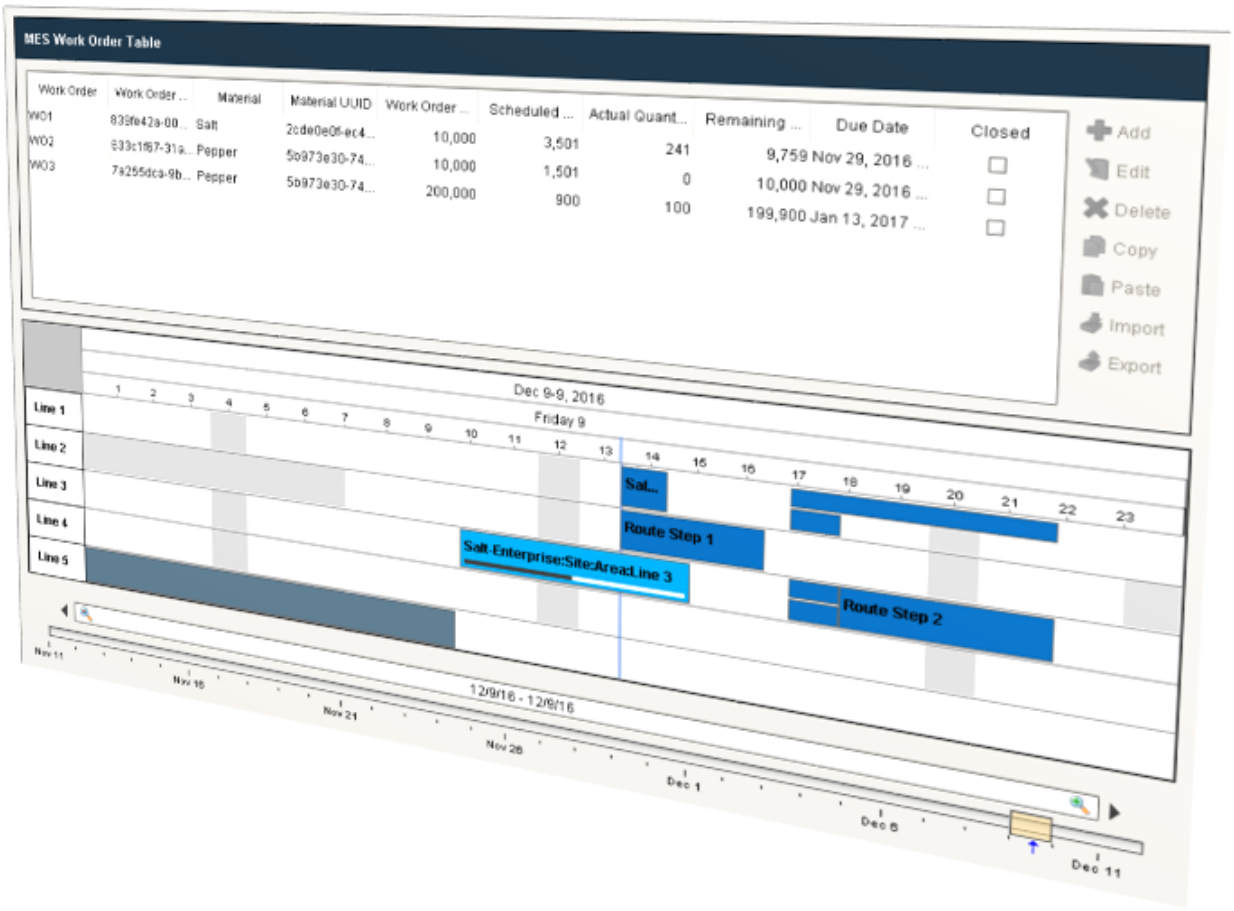
The Track & Trace module can be used for detailed production scheduling on any type of manufacturing operation. Refer to Operations Scheduling for more details.

OEE 2.0 Module

The OEE 2.0 module can also be used for detailed production scheduling of OEE production runs, maintenance and cleaning operations. Refer to Operations Scheduling for more details.

The MES Scheduler is common between both Track & Trace and OEE 2.0.

MES Scheduler



The MES Scheduler provides finite scheduling functionality that seamlessly integrates with the OEE 2.0 and Track & Trace modules. When combined with Ignition, these scheduling features allow operations to easily adapt to last minute or frequent changes that commonly occur in production environments. This is accomplished by monitoring production in real-time, handling delays, production routes, scheduling changes, notification of production priority changes and more.

Most Manufacturers rely on ERP and Inventory Management Systems to handle the complex process of inventory planning, high level Customer Order scheduling, routing, transportation logistics and accounting. However when it comes to the detailed planning and scheduling of shift personnel, production lines and maintenance activities, this tends to occur at the MES level. If scheduling information and production performance data is stored in separate systems, whether in spreadsheets or stand-alone scheduling software, that cannot be accessed and combined, we're missing the opportunity to provide powerful insight into operational activities and production line utilization.

The Scheduler provides finite scheduling functionality at the MES layer that allows operations, maintenance and planners to create a detailed web based schedule that can pull work orders directly from ERP, allowing them to be modified at the MES level to account for last minute changes, and shared with everyone within the organization. With our direct connection to actual production line status and counts, the schedule can provide real-time status monitoring of actual vs scheduled production, automatic handling of delays and updates of inventory consumption and order fulfillment data back to ERP, as well as providing schedule adherence analytics to help drive continuous improvement initiatives.

With flexibility in mind, work orders and production schedule entries can be created in the following ways:

- From Customer Orders or Schedule Entries in ERP or other scheduling software
- Manually created using the Work Order Table and Line Schedule View components
- Dynamically generated using scripting
- Imported from other sources

Features

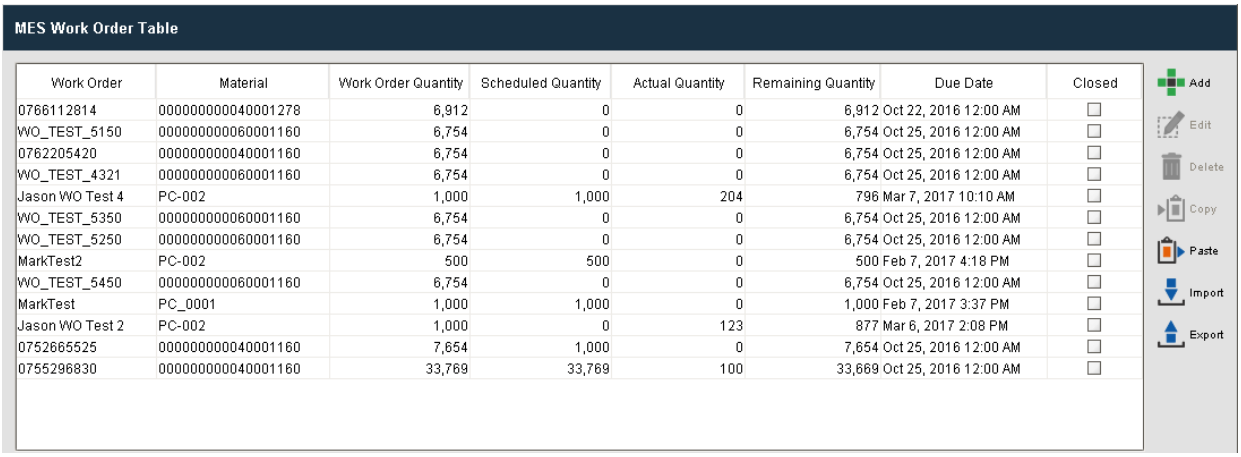
- **Stand-Alone Scheduling or ERP Work Order Integration**
- **Simple 'Drag & Drop' Work Order Scheduling**
- **Production Routing**
- **Shift Scheduling**
- **Automatic Adjustment for Production Delays**
- **Finite Scheduling estimates completion date based on constraints**
- **Real-time production progress**
- **Seamless integration with all other Sepasoft MES modules**
- **Visual Scheduler for easy communication of schedule between departments**
- **Overlapping schedule entries**
- **Auto Extend Scheduled Run (when production is behind schedule)**

In This Section

Creating and Managing Work Orders

Work Order Table

In a manufacturing environment, Work Orders are created from customer orders to define a desired quantity of a certain product and a due date. Other information such as priority and status may also be attached to a Work Order. The Work Order Table component allows for the creation and editing of Work Orders. Work Orders can then be scheduled across production lines and the quantity scheduled, produced and remaining can be tracked.



In this Page

Creating Work Orders

The MES Work Order Table component is used to create and edit work orders.

Adding, Editing and Deleting Work Orders

Work Orders can be created at the **Material Definition** level.

- **Add** - Click on the 

button and provide a name for the work order. Select the Material Definition, assign the quantity and choose the due date.
- **Edit -** Select the work order you want to edit and click on the 

button. Edit the settings and click **Save**.
- **Delete** - Select the work order and click on the 

button.

Copying and Pasting a Work Order

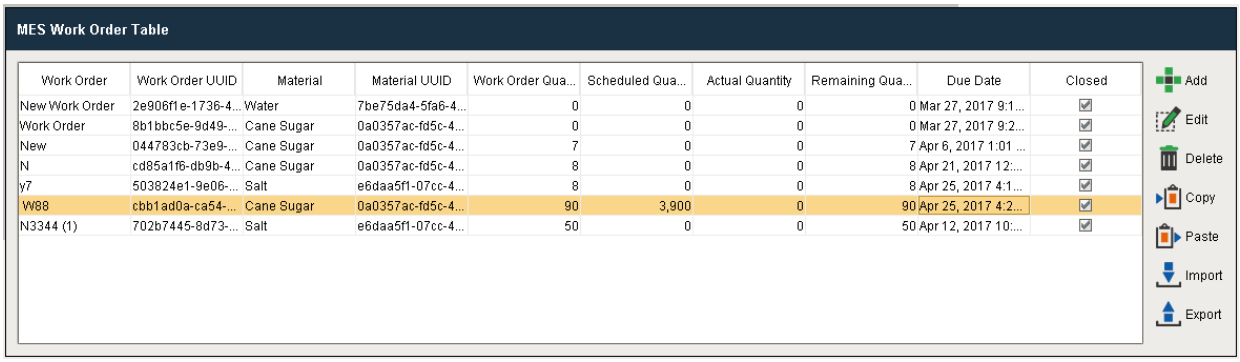
- Select the Work order to be copied, click **Copy** and then click **Paste**.
- The pasted Work Order will be shown at the bottommost row.

Exporting Work Order

- Click **Export**. All the work orders will be exported at once.
- When the save window appears, name the file to be exported and Click **Save**.

Importing Work Order

- Click **Import**.



- Use the File open dialog box to select the xml file to be imported and Click **Open**.

Expand to see an example format of the XML file

```
<?xml version="1.0"?>
```

```
-<MESObjectList>
-<MESObject MESObjectType="WorkOrder">
<CoreProperty name="UUID">2e906f1e-1736-4e02-8586-
6b7e4e7e17fc</CoreProperty>
<CoreProperty name="Name">New Work Order</CoreProperty>
<CoreProperty name="Enabled">true</CoreProperty>
<CoreProperty name="Creator">Unknown</CoreProperty>
<CoreProperty name="MaterialRef">Material Definition,
Water</CoreProperty>
<CoreProperty name="MaterialRefUUID">7be75da4-5fa6-4b9b-
abae-087e00cb1971</CoreProperty>
<CoreProperty name="MaterialRefType">MaterialDef</CoreProper
ty>
<CoreProperty name="WorkOrderQuantity">0.0</CoreProperty>
<CoreProperty name="WorkOrderActualQuantity">0.0</CoreProper
ty>
<CoreProperty name="WorkOrderScheduleQuantity">0.0</CoreProp
erty>
<CoreProperty name="WorkOrderRemainingQuantity">0.0</CorePro
perty>
<CoreProperty name="WorkOrderDueDate">2017-03-27 09:18:35</C
oreProperty>
<CoreProperty name="WorkOrderClosed">true</CoreProperty>
</MESObject>
-<MESObject MESObjectType="WorkOrder">
<CoreProperty name="UUID">702b7445-8d73-4946-b990-
ce79188ed5fa</CoreProperty>
<CoreProperty name="Name">N3344 (1)</CoreProperty>
<CoreProperty name="Enabled">true</CoreProperty>
<CoreProperty name="Creator">Unknown</CoreProperty>
<CoreProperty name="MaterialRef">Material Definition, Salt</
CoreProperty>
<CoreProperty name="MaterialRefUUID">e6daa5f1-07cc-4863-
b3b3-0d560c514c63</CoreProperty>
<CoreProperty name="MaterialRefType">MaterialDef</CoreProper
ty>
<CoreProperty name="WorkOrderQuantity">50.0</CoreProperty>
<CoreProperty name="WorkOrderActualQuantity">0.0</CoreProper
ty>
<CoreProperty name="WorkOrderScheduleQuantity">0.0</CoreProp
erty>
<CoreProperty name="WorkOrderRemainingQuantity">50.0</CorePr
operty>
<CoreProperty name="WorkOrderDueDate">2017-04-12 10:49:49</C
oreProperty>
<CoreProperty name="WorkOrderClosed">false</CoreProperty>
</MESObject>
-<MESObject MESObjectType="WorkOrder">
<CoreProperty name="UUID">cd85a1f6-db9b-4220-88ae-
001e8745a185</CoreProperty>
<CoreProperty name="Name">N</CoreProperty>
```


```
<CoreProperty name="Enabled">true</CoreProperty>
<CoreProperty name="Creator">Unknown</CoreProperty>
<CoreProperty name="MaterialRef">Material Definition, Cane
Sugar</CoreProperty>
<CoreProperty name="MaterialRefUUID">0a0357ac-fd5c-454f-
91ff-119aadaa5014</CoreProperty>
<CoreProperty name="MaterialRefType">MaterialDef</CoreProper
ty>
<CoreProperty name="WorkOrderQuantity">8.0</CoreProperty>
<CoreProperty name="WorkOrderActualQuantity">0.0</CoreProper
ty>
<CoreProperty name="WorkOrderScheduleQuantity">0.0</CoreProp
erty>
<CoreProperty name="WorkOrderRemainingQuantity">8.0</CorePro
perty>
<CoreProperty name="WorkOrderDueDate">2017-04-21 12:49:50</C
oreProperty>
<CoreProperty name="WorkOrderClosed">true</CoreProperty>
</MESObject>
-<MESObject MESObjectType="WorkOrder">
<CoreProperty name="UUID">044783cb-73e9-4c2b-abf3-
13ee0158d40c</CoreProperty>
<CoreProperty name="Name">New </CoreProperty>
<CoreProperty name="Enabled">true</CoreProperty>
<CoreProperty name="Creator">Unknown</CoreProperty>
<CoreProperty name="MaterialRef">Material Definition, Cane
Sugar</CoreProperty>
<CoreProperty name="MaterialRefUUID">0a0357ac-fd5c-454f-
91ff-119aadaa5014</CoreProperty>
<CoreProperty name="MaterialRefType">MaterialDef</CoreProper
ty>
<CoreProperty name="WorkOrderQuantity">7.0</CoreProperty>
<CoreProperty name="WorkOrderActualQuantity">0.0</CoreProper
ty>
<CoreProperty name="WorkOrderScheduleQuantity">0.0</CoreProp
erty>
<CoreProperty name="WorkOrderRemainingQuantity">7.0</CorePro
perty>
<CoreProperty name="WorkOrderDueDate">2017-04-06 13:01:16</C
oreProperty>
<CoreProperty name="WorkOrderClosed">true</CoreProperty>
</MESObject>
-<MESObject MESObjectType="WorkOrder">
<CoreProperty name="UUID">8b1bbc5e-9d49-4fe5-a967-
6c66d4bc90b7</CoreProperty>
<CoreProperty name="Name">Work Order</CoreProperty>
<CoreProperty name="Enabled">true</CoreProperty>
<CoreProperty name="Creator">Unknown</CoreProperty>
<CoreProperty name="MaterialRef">Material Definition, Cane
Sugar</CoreProperty>
```

```
<CoreProperty name="MaterialRefUUID">0a0357ac-fd5c-454f-
91ff-119aadaa5014</CoreProperty>
<CoreProperty name="MaterialRefType">MaterialDef</CoreProper
ty>
<CoreProperty name="WorkOrderQuantity">0.0</CoreProperty>
<CoreProperty name="WorkOrderActualQuantity">0.0</CoreProper
ty>
<CoreProperty name="WorkOrderScheduleQuantity">0.0</CoreProp
erty>
<CoreProperty name="WorkOrderRemainingQuantity">0.0</CorePro
perty>
<CoreProperty name="WorkOrderDueDate">2017-03-27 09:21:19</C
oreProperty>
<CoreProperty name="WorkOrderClosed">true</CoreProperty>
</MESObject>
-<MESObject MESObjectType="WorkOrder">
<CoreProperty name="UUID">503824e1-9e06-4b12-982d-
69f1d5a3bd5a</CoreProperty>
<CoreProperty name="Name">y7</CoreProperty>
<CoreProperty name="Enabled">true</CoreProperty>
<CoreProperty name="Creator">Unknown</CoreProperty>
<CoreProperty name="MaterialRef">Material Definition, Salt</
CoreProperty>
<CoreProperty name="MaterialRefUUID">e6daa5f1-07cc-4863-
b3b3-0d560c514c63</CoreProperty>
<CoreProperty name="MaterialRefType">MaterialDef</CoreProper
ty>
<CoreProperty name="WorkOrderQuantity">8.0</CoreProperty>
<CoreProperty name="WorkOrderActualQuantity">0.0</CoreProper
ty>
<CoreProperty name="WorkOrderScheduleQuantity">0.0</CoreProp
erty>
<CoreProperty name="WorkOrderRemainingQuantity">8.0</CorePro
perty>
<CoreProperty name="WorkOrderDueDate">2017-04-25 16:14:23</C
oreProperty>
<CoreProperty name="WorkOrderClosed">true</CoreProperty>
</MESObject>
-<MESObject MESObjectType="WorkOrder">
<CoreProperty name="UUID">cbb1ad0a-ca54-4397-a287-
d6eb0e92721c</CoreProperty>
<CoreProperty name="Name"> W88</CoreProperty>
<CoreProperty name="Enabled">true</CoreProperty>
<CoreProperty name="Creator">Unknown</CoreProperty>
<CoreProperty name="MaterialRef">Material Definition, Cane
Sugar</CoreProperty>
<CoreProperty name="MaterialRefUUID">0a0357ac-fd5c-454f-
91ff-119aadaa5014</CoreProperty>
<CoreProperty name="MaterialRefType">MaterialDef</CoreProper
ty>
<CoreProperty name="WorkOrderQuantity">90.0</CoreProperty>
```

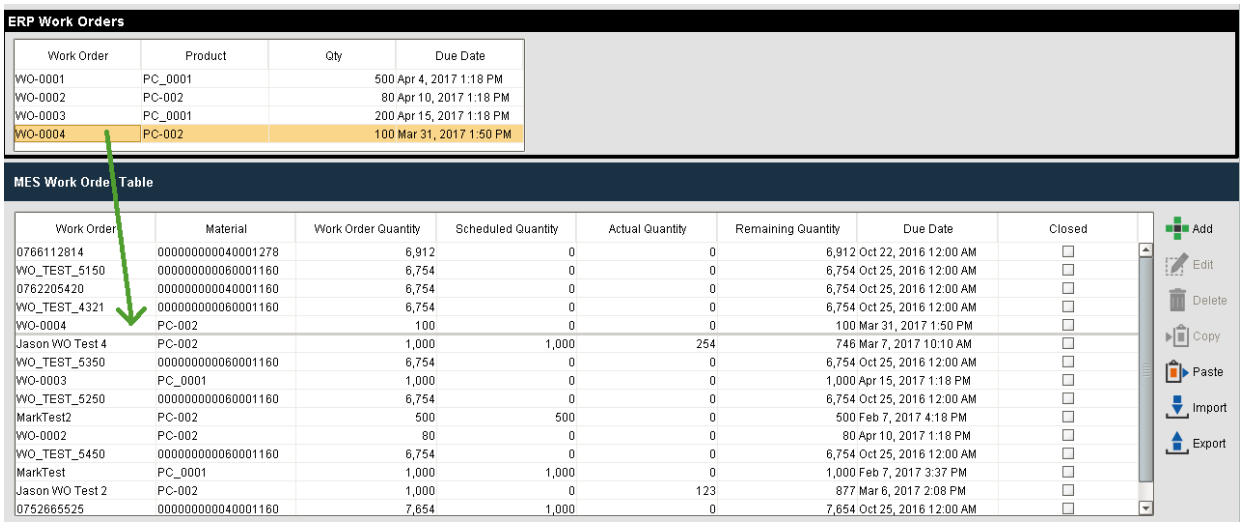
```
<CoreProperty name="WorkOrderActualQuantity">0.0</CoreProperty>
<CoreProperty name="WorkOrderScheduleQuantity">0.0</CoreProperty>
<CoreProperty name="WorkOrderRemainingQuantity">0.0</CoreProperty>
<CoreProperty name="WorkOrderDueDate">2017-04-25 16:26:38</CoreProperty>
<CoreProperty name="WorkOrderClosed">true</CoreProperty>
</MESObject>
</MESObjectList>
```

Managing Work Orders

Sepasoft's MES provides a number of components and scripting functions to allow for the management and scheduling of Work Orders.

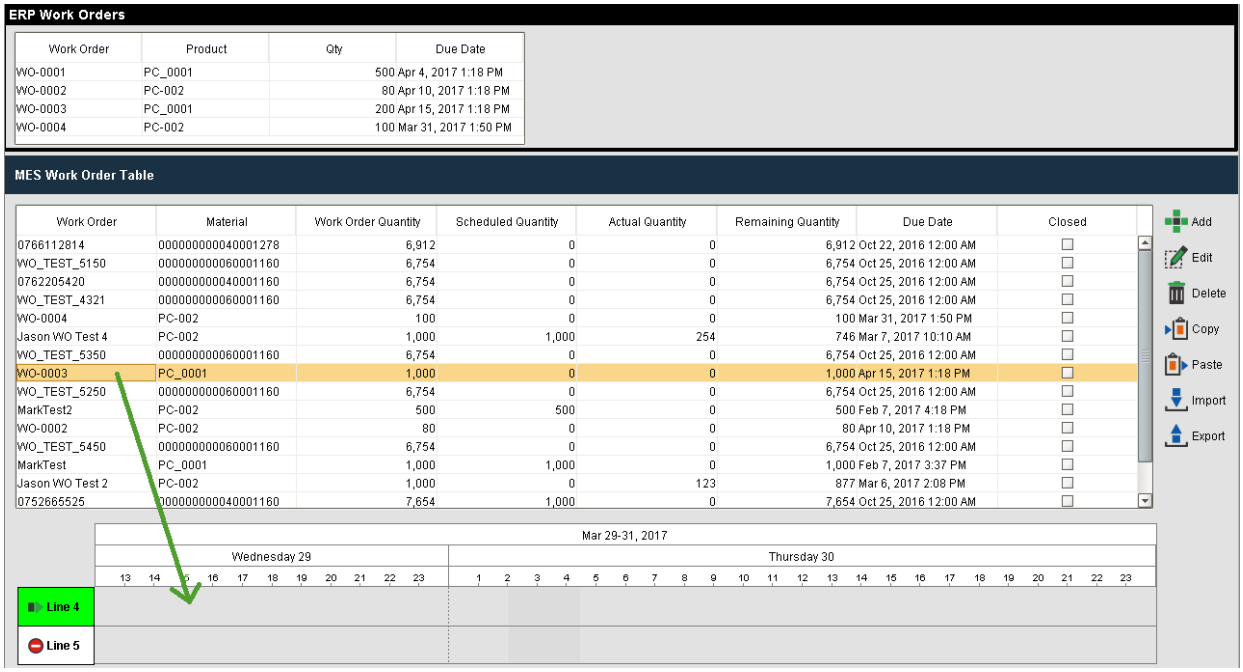
Drag and Drop Work Order Creation

The MES Work Order Table component allows for dragging and dropping rows from a power table onto itself. You could create a power table that pulls work order from ERP and allows a user to drag Work Orders over. In order to perform drag and drop, you must enable the **Row Dragging Enabled property** of the power table .



Drag and Drop Work Order Scheduling

The MES Work Order Table component also allows for dragging and dropping a Work Order onto the MES Schedule View Component.



Scripting Functions

Along with the Work Order Table component, scripting functions are also provided in the system.mes.workorder space for managing work orders. This provides an automated method for pulling Work Orders out of an ERP system and creating the associated Work order objects in the MES layer.

The following scripting functions are available

- system.mes.workorder.createMESWorkOrder
- system.mes.workorder.getMESWorkOrder
- system.mes.workorder.getMESWorkOrders
- system.mes.workorder.deleteMESWorkOrders
- system.mes.workorder.createMESWorkOrderFilter
- system.mes.workorder.getMESWorkOrderObjectLinkList
- system.mes.workorder.saveMESWorkOrder

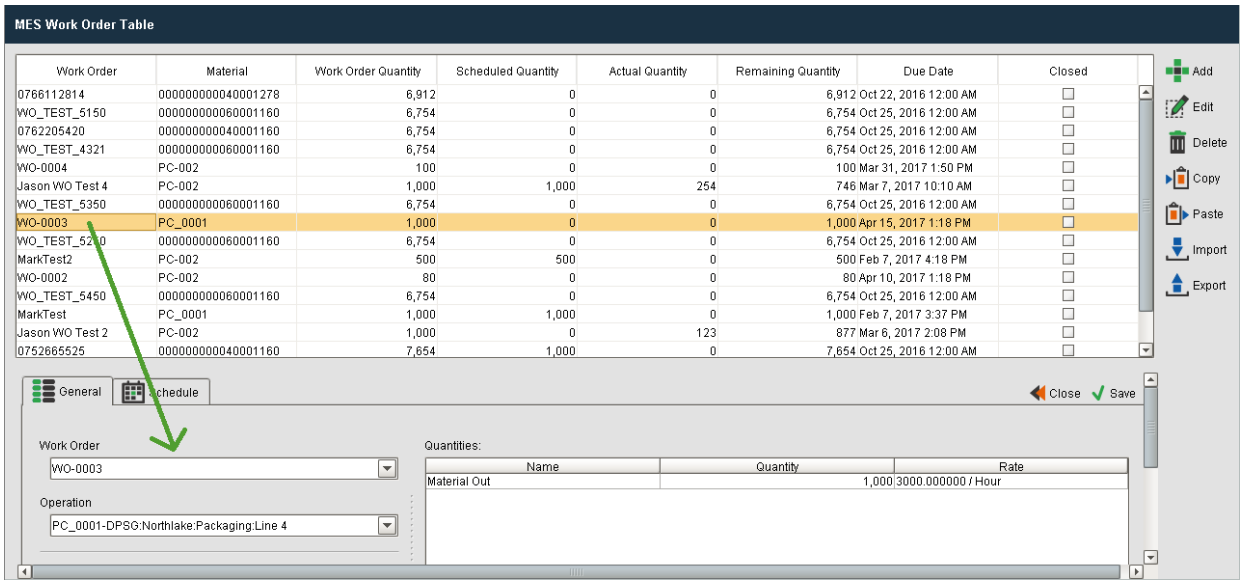
Operations Scheduling

Sepasoft's MES provides the ability to schedule any type of operation, whether it is a Work Order production run, Maintenance Work Order, cleaning operation, new production introduction, testing etc. Any type of operation can be created using the MES Object Editor component or through scripting. These scheduled operations can then be used by the Track & Trace module for lot tracking, material consumption and WIP inventory, and by the OEE module for order fulfillment and downtime tracking.

Run Scheduling

Work Orders

When the MES Work Order Table component is used, Work Orders can be dragged from the Work Order table and dropped onto the MES Schedule View component. For OEE runs, the OEE Material Manager component must be used to create the Material Root Material Definitions and to associate these Material Definitions with the Production Line(s).

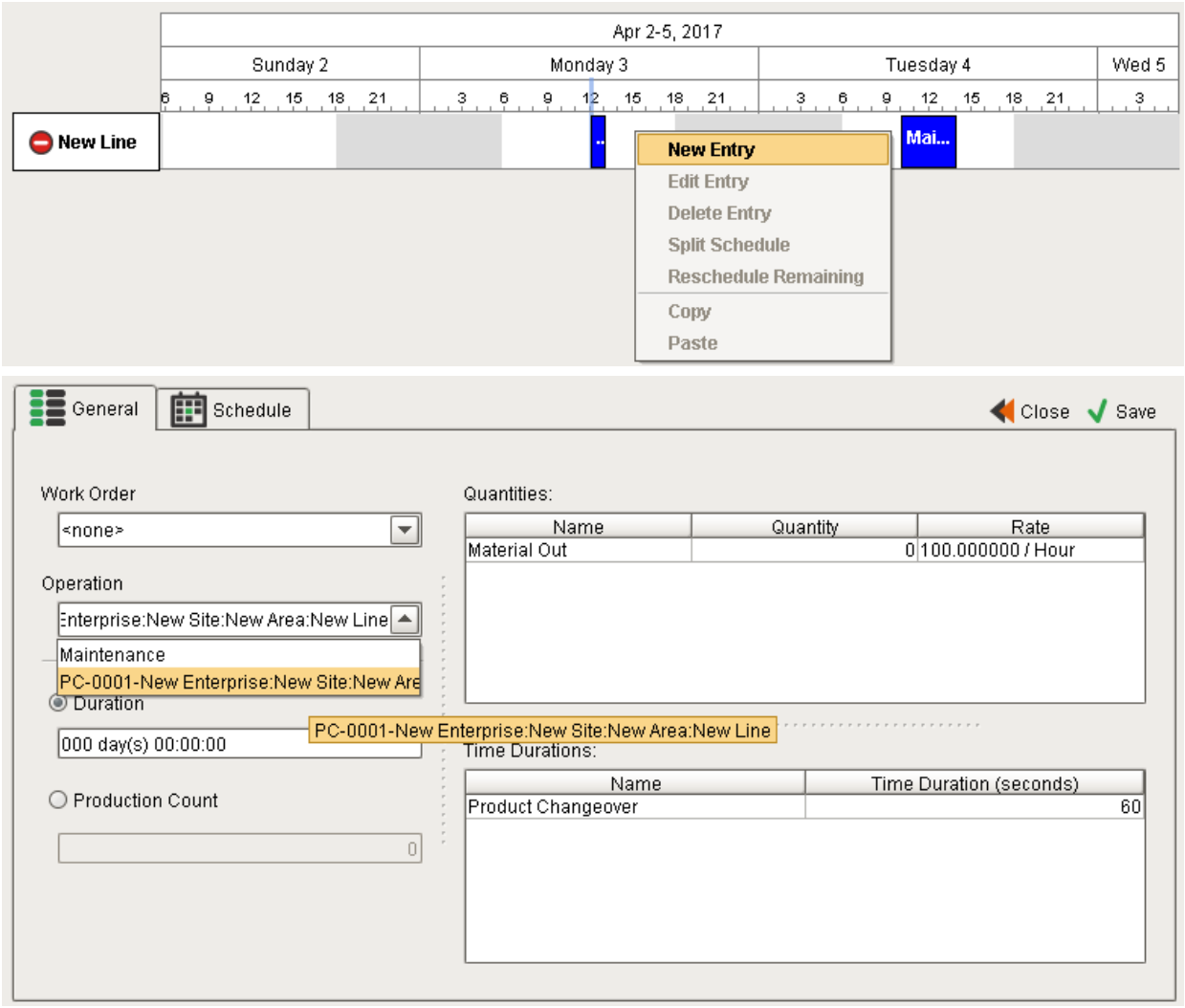


MES Work Order Table and MES Schedule View component

Product Codes

When scheduling a Product Code that is not part of a Work Order, the MES Schedule View component can be used to create the scheduled run.

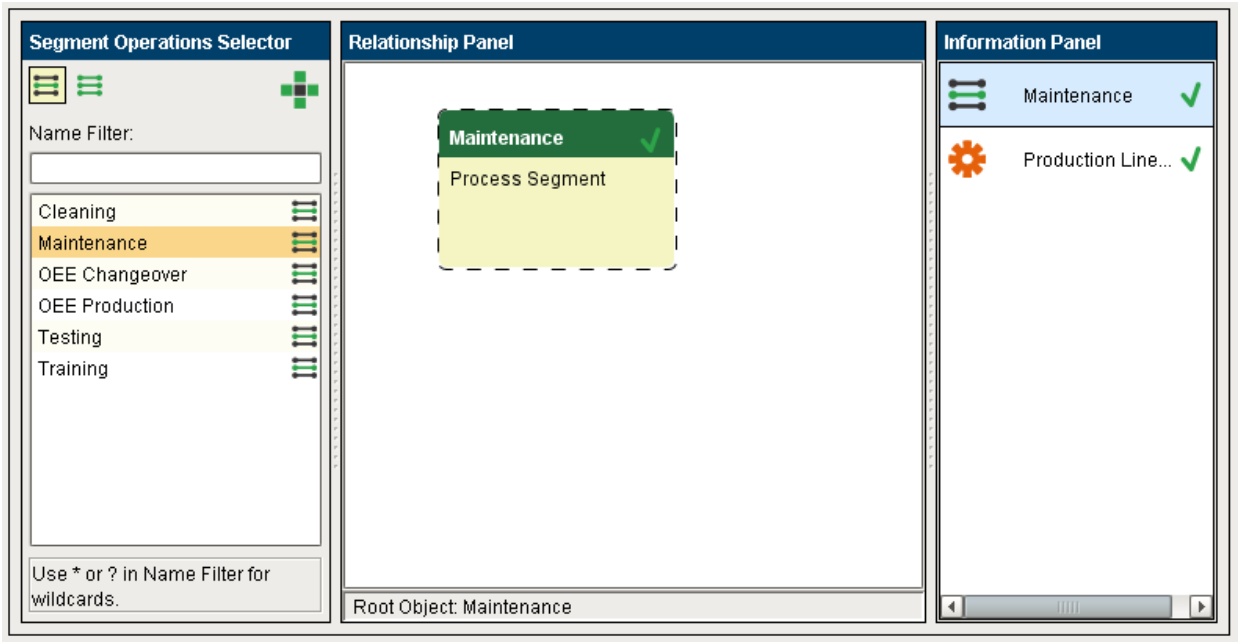
- Right click on the desired line at the time you wish to schedule the run and select **New Entry**.
- Select <**none>** for the Work Order and select the product code operation you wish to schedule.
- Seelct a duration or a Production Count for the run and **Save**.



MES Schedule View component

Maintenance and Other Operations Scheduling

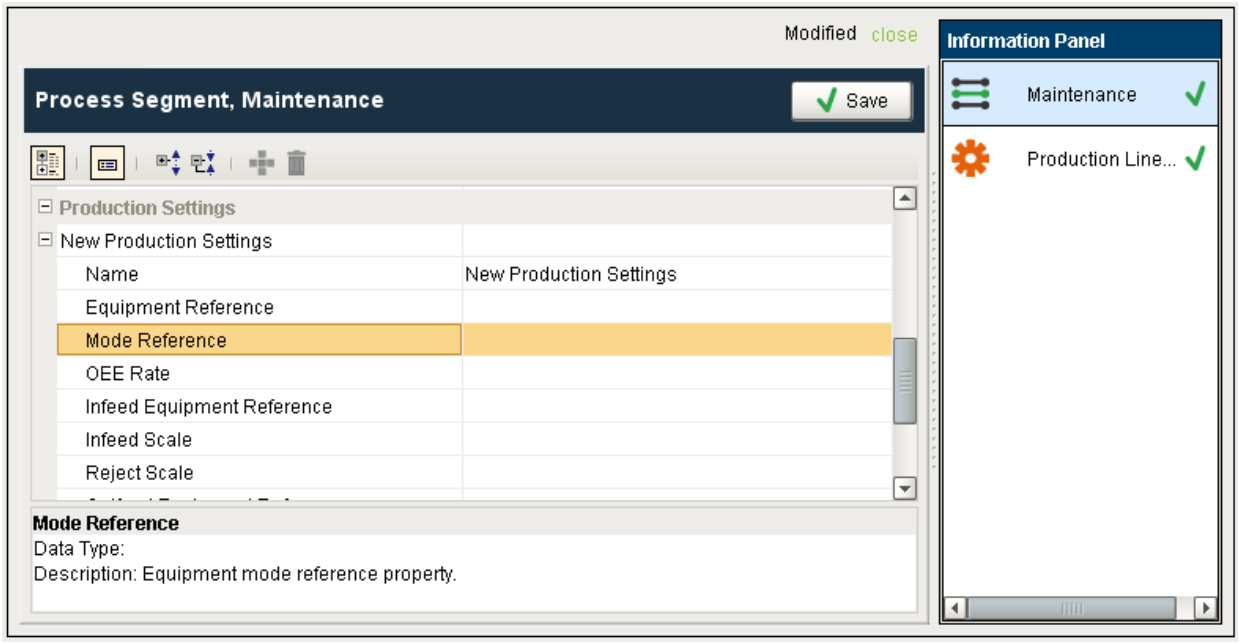
Any type of operation can be created and scheduled. The operations can be created using the MES Object Editor or through scripting functions. These operations do not need to have Material Definitions associated with them unless you want to track consumables, but they do require equipment to be associated with them. If you want to create a maintenance operation that can be scheduled on any line, simply associate the process segment with an equipment class that includes all production lines.



MES Object Editor component

The process segment has associated Production settings with it. Select the mode you want to consider the equipment to be in when this operation is active. This will allow you to use analysis to return the equipment mode to provide an equipment utilization chart.

You can then use the MES Schedule View component to schedule these operations by selecting **<none>** for Work Order and selecting the appropriate Operation.



Process Segment Production Settings

ERP Integration

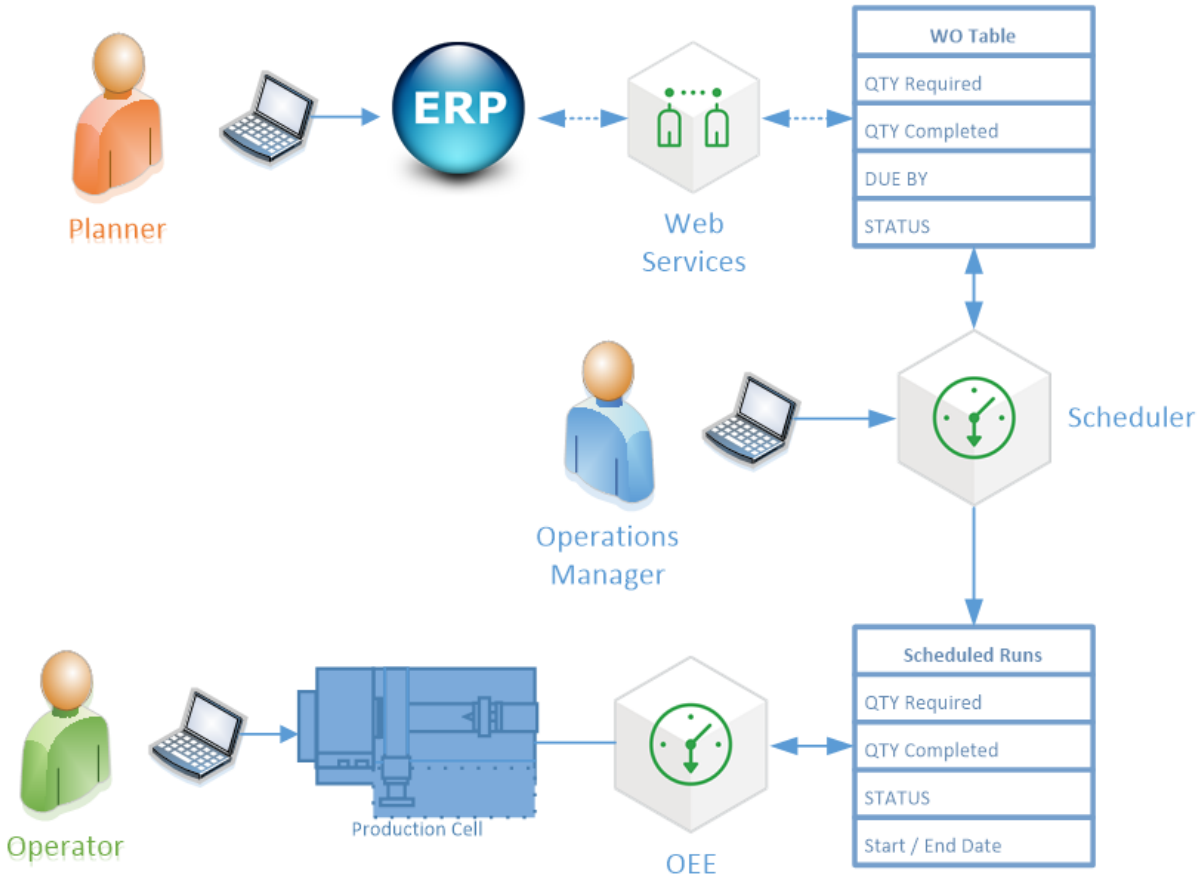
Although the Sepasoft MES Modules can be used to create a stand-alone application, allowing you to create product codes, work orders, routes and schedules, it is also possible to integrate your MES solution with a third party ERP or scheduling system to allow for the bi-directional flow of information between ERP and MES. The following section provides a number of possible solutions based on the flow and type of data you can expect to pass between MES and other third party systems. actual implementation will be depended on the interface method employed. For more information on the types of interfaces that are available, please refer to the knowledge base article on 'Creating an Interface Exchange between ERP and MES'.

ERP Work Order Scheduling

1. ERP Customer Orders or scheduled Process Orders are automatically pulled down from ERP to the MES layer using Web Services or middleware tables

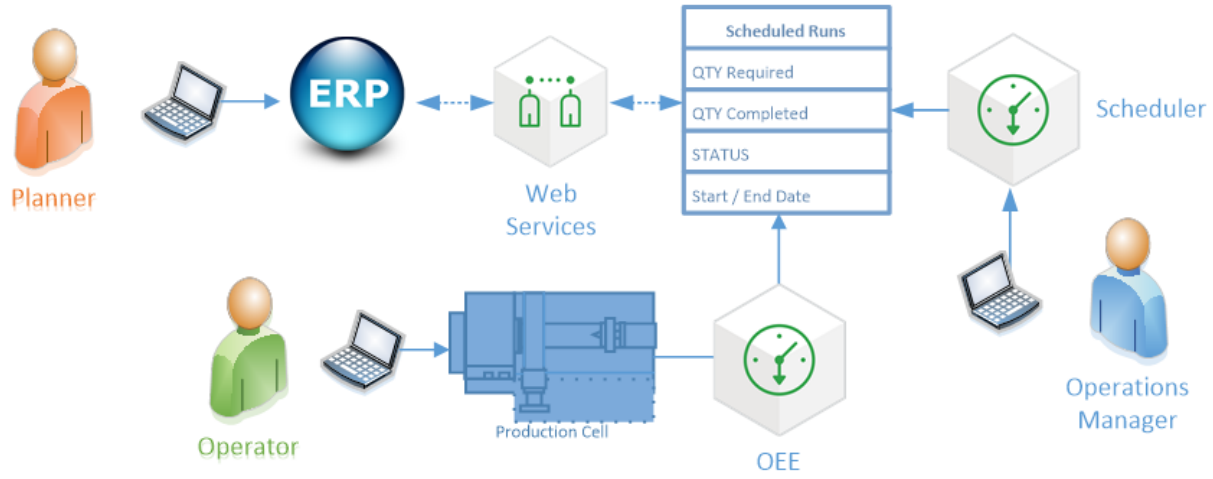
1. Operations Management schedules production based on Work Orders or scheduled runs are dynamically created through scripting
2. Operator selects a Work Order run or starts an un-scheduled production run against a Work Order. Alternatively, production line state change event triggers a production run start

1. Production counts are automatically captured and available at the ERP level directly or after approval by Operations Management



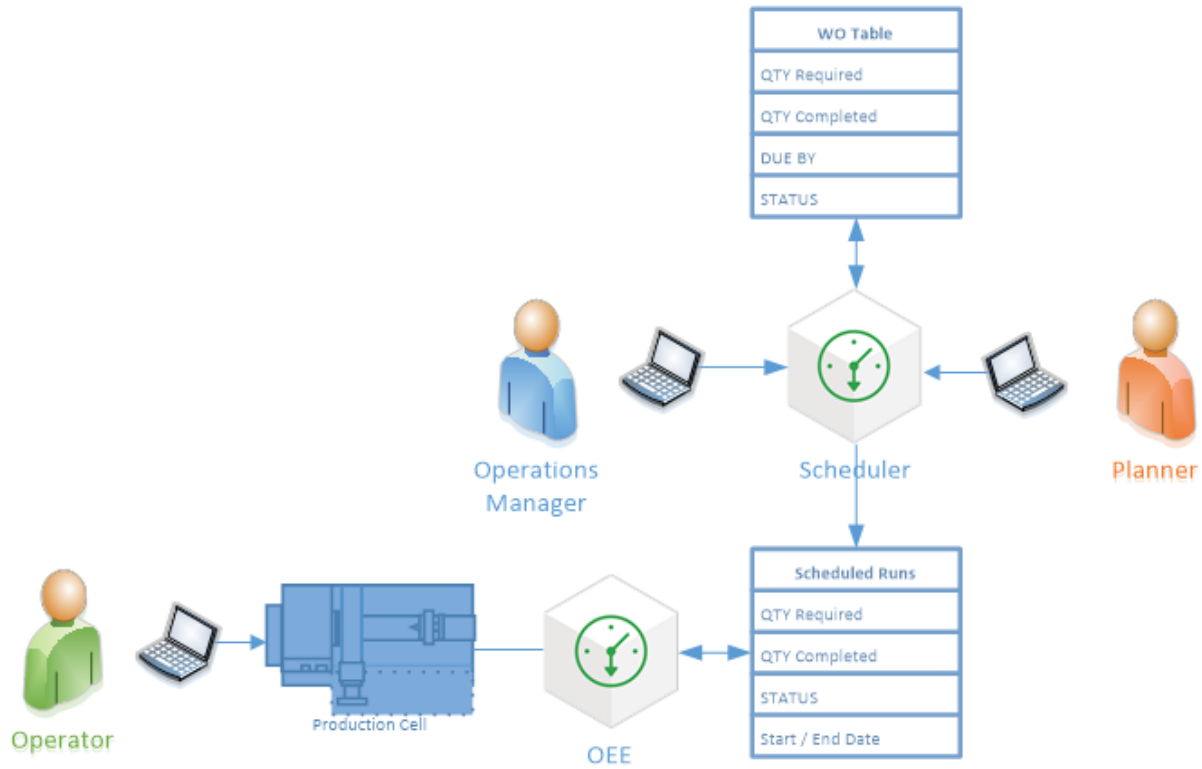
ERP Production Run Scheduling

- Customer Orders are scheduled in ERP by Planner and downloaded to MES as Scheduled Production Runs
- Operations Management can adjust schedules as necessary. Status updates can be sent back to ERP
- Operator selects a scheduled production run. Alternatively, production line status change event triggers a production run start



Stand-Alone Work Order Scheduling

- Planner creates Work Orders in MES Layer
- Operation Management schedules Production runs
- Operator selects a scheduled production run or starts an un-scheduled production run. Alternatively, production line status change event triggers a production run start
- Production Counts are automatically captured



6.8.4 Production Dispatching

ISA-95 defines production dispatching as '...*the collection of activities that manage the flow of production by dispatching production to equipment and personnel'.* The Sepasoft MES solution provides the OEE 2.0 and Track & Trace modules that can handle the needs of your production dispatching system. Most aspects of Production Dispatching are handled through the MES Scheduler in terms of issuing work to resources, updating schedules based on unforeseen events and assigning the material, resources and equipment to be used.

Track & Trace Module

The Track & Trace module can be used for production dispatching on any type of operation. Refer to the Operations Scheduling for more details.

OEE 2.0 Module

The OEE 2.0 module can also be used for production dispatching on operations for which OEE downtime tracking is required. Refer to Operations Scheduling for more details.

6.8.5 Production Execution Management

ISA-95 defines production execution management as '... *the collection of activities that direct the performance of work, as specified by the contents of the production dispatch list elements. The production execution management activity includes selecting, starting and moving those units of work (for example lots, sublots, or batches) through the appropriate sequence of operations to physically produce the product'.*

The Sepasoft MES solution provides the OEE 2.0 and Track & Trace modules that can be used to build your production execution management system. Please refer to OEE 2.0 Module and Track & Trace Overview for more details.

Track & Trace Module

The Track & Trace module can be used for production execution management on any type of operation and handles sublots and operation sequencing. Refer to the Operations Scheduling for more details.

OEE 2.0 Module

The OEE 2.0 module can also be used for production execution management on operations for which OEE downtime tracking is required, but does not handle sublots or operation sequencing. Refer to Production Order Execution for more details.

6.8.6 Production Data Collection

ISA-95 defines production data collection as...

'...the collection of activities that gather, compile and manage production data for specific work processes or specific production requests. Manufacturing control systems generally deal with process information such as quantities (weight, units, etc.) and associated properties (rates, temperatures, etc.) and with equipment information such as controller, sensor, and actuator statuses. The managed data may include sensor readings, equipment states, event data, operator-entered data, transaction data, operator actions, messages, calculation results from models, and other data of importance in the making of a product. The data collection is inherently time or event based, with time or event data added to give context to the collected information.'

The Sepasoft MES solution provides the OEE 2.0 and Track & Trace modules that can be used to build your production data collection system, in conjunction with the Recipe Management, SPC, Instrument Interface, Barcode Scanner and Web Services modules, as well as the Ignition historian for associated properties.

Track & Trace Module

The Track & Trace module can be used for production data collection on any type of operation providing lot tracking, WIP inventory, raw material consumption, order fulfillment and any other data values logged during operation execution. Refer to the Track & Trace Overview for more details.

OEE 2.0 Module

The OEE 2.0 module can also be used for production data collection on OEE production runs providing OEE metrics, order fulfillment, schedule adherence and any other data values logged during operation execution . Refer to the OEE 2.0 Module for more details.

Recipe Management Module

The Recipe Management provides recipe and actual machine setpoints including variance for an operation. This data can be coupled with the operational data from Track & Trace and OEE 2.0 to provide insight into the machine settings and recipe parameters used during an operation. Refer to Recipe Management Overview for more information on using this module for your production data collection system.

SPC Module

The SPC module provides real-time statistical analysis of process and production variables during an operation. This data can be coupled with the operational data from Track & Trace and OEE 2.0 to provide insight on the stability of process variables during an operation. Refer to SPC Module Overview for more information on using this module for your production data collection system.

Instrument Interface Module

The Instrument Interface modules provide a means of pulling data out of inspection equipment and process cells that can provide a flat file or rs-232 communication source for passing information to other systems. Refer to Instrument Interface Overview for more information on using this module for your production data collection system.

Barcode Scanner Module

The barcode scanner module provides a method for simplifying data entry from operators scanning incoming material lot ids, tool dies sets and other information from a barcode. Refer to Barcode Scanner Module Overview for more information on using this module for your production data collection system.

Web Services Module

The Web Services module allows you to connect your MES to any other information system or data source that provides a SOAP or RESTful API. Any data you wish to store during a production operation can be pulled from another system using this interface and stored along with the production run information. Refer to Web Services Module for more information on using this module for your production data collection system.

6.8.7 Production Tracking

ISA-95 defines production tracking as...

'...the collection of activities that prepare the production response for Level 4. This includes summarizing and reporting information about personnel and equipment actually used to produce product, material consumed, material produced, and other relevant production data such as costs and performance analysis results. Production tracking also provides information to detailed production scheduling and Level 4 scheduling activities so schedules can be updated based on current conditions'.

The Sepasoft MES solution provides the OEE 2.0 and Track & Trace modules that can be used to build your production tracking system, in conjunction with the Recipe Management, SPC, Instrument Interface, Barcode Scanner and Web Services modules.

Track & Trace Module

The Track & Trace module can be used for production tracking on any type of operation providing lot tracking, WIP inventory, raw material consumption, personnel and equipment used, scheduling activities, order fulfillment and any other data values logged during operation execution. Refer to the Track & Trace Overview for more details.

OEE 2.0 Module

The OEE 2.0 module can also be used for production data collection on OEE production runs providing OEE metrics, order fulfillment, schedule activities, and any other data values logged during operation execution . Refer to the OEE 2.0 Module for more details.

Recipe Management Module

The Recipe Management provides recipe and actual machine setpoints including variance for an operation. This data can be coupled with the operational data from Track & Trace and OEE 2.0 to track machine settings and recipe parameters used during an operation. Refer to Recipe Management Overview for more information on using this module as part of for your production tracking system.

SPC Module

The SPC module provides real-time statistical analysis of process and production variables during an operation. This data can be coupled with the operational data from Track & Trace and OEE 2.0 to provide insight on the stability of process variables during an operation. Refer to SPC Module Overview for more information on using this module as part of for your production tracking system.

Instrument Interface Module

The Instrument Interface modules provide a means of pulling data out of inspection equipment and process cells that can provide a flat file or rs-232 communication source for passing information to other systems. Refer to Instrument Interface Overview for more information on using this module as part of for your production tracking system.

Barcode Scanner Module

The barcode scanner module provides a method for simplifying data entry from operators scanning incoming material lot ids, tool dies sets and other information from a barcode. Refer to Barcode Scanner Module Overview for more information on using this module as part of for your production tracking system.

Web Services Module

The Web Services module allows you to connect your MES to any other information system or data source that provides a SOAP or RESTful API. Any data you wish to track during a production operation can be pulled from another system using this interface and stored along with the production run information. Refer to Web Services Module for more information on using this module as part of for your production tracking system.

6.8.8 Production Performance Analysis

ISA-95 defines production performance analysis as...

'...the collection of activities that analyze and report performance information to business systems. This would include analysis of information of production unit cycle times, resource utilization, equipment utilization, equipment performance, procedure efficiencies, and production variability.'

The Sepasoft MES solution provides the OEE 2.0 module that can be used to build your production performance analysis system, in conjunction with the Track & Trace, Recipe Management, SPC, Instrument Interface, Barcode Scanner and Web Services modules, as well as the Ignition historian for associated properties.

OEE 2.0 Module

The OEE 2.0 module can also be used for production performance analysis on OEE production runs providing OEE metrics, order fulfillment, schedule activities, and any other data values logged during operation execution. Refer to the OEE 2.0 Module for more details.

Track & Trace Module

The Track & Trace module provides lot tracking, WIP inventory, raw material consumption, personnel and equipment used, scheduling activities, order fulfillment and any other data values logged during operation execution. Refer to the Track & Trace Overview for more details.

Recipe Management Module

The Recipe Management provides recipe and actual machine setpoints including variance for an operation. This data can be coupled with the operational data from Track & Trace and OEE 2.0 to track machine settings and recipe parameters used during an operation. Refer to Recipe Management Overview for more information on using this module as part of for your performance analysis system.

SPC Module

The SPC module provides real-time statistical analysis of process and production variables during an operation. This data can be coupled with the operational data from Track & Trace and OEE 2.0 to provide insight on the stability of process variables during an operation. Refer to SPC Module Overview for more information on using this module as part of for your performance analysis system.

6.9 Maintenance Operations Management

ISA-95 defines maintenance operations management as ...

'... the collection of activities which coordinate, direct, and track the functions that maintain the equipment, tools and related assets to ensure their availability for manufacturing and ensure scheduling for reactive, periodic, preventive, or proactive maintenance.'

Today, Sepasoft's MES solution provides the ability for combining maintenance operations and production scheduling all in one system, allowing for easy visual communication between departments. Ensuring only certified personnel can accomplish certain tasks can be handled through the Track & Trace production control feature, and the OEE 2.0 module provides MTBF metrics as well as downtime tracking and pareto analysis. Refer to OEE 2.0 Module and Track & Trace Overview for more details.

The next scheduled module release from Sepasoft will be the Maintenance Module!

6.10 Quality Operations Management

ISA-95 defines quality operations management as ...

'...the collection of activities which coordinate, direct, and track the functions that measure and report on quality. The broad scope of quality operations management includes both quality operations and the management of those operations in order to ensure the quality of

intermediate and final products. Quality operations management may include: a) Testing and verifying the quality of materials (raw, final, and intermediate). b) Measuring and reporting the capability of the equipment to meet quality goals. c) Certifying product quality. d) Setting standards for quality. e) Setting standards for quality personnel certification and training. f) Setting standards for control of quality.

Sepasoft's MES solution provides the SPC module for realizing ISA-95's goal for Quality Operations Management. Refer to SPC Module Overview for more details.

6.11 Inventory Operations Management

ISA-95 defines inventory operations management as ...

'...Managing and tracking the inventory of product and/or material. b) Performing periodic and /or on-demand inventory cycle counts. c) Managing the transfer of material between work centers. d) Measuring and reporting on inventory and material transfer capabilities. e) Coordinating and controlling the personnel and equipment used in material transfer. f) Directing and monitoring the transfer of material to and from production, quality, or maintenance. g) Reporting on inventory to production, quality, maintenance operations management, and/or Level 4 activities. h) Routing raw material to and from storage. i) Identifying pack out schedules. j) Staging and monitoring the movement of material in storage.

Sepasoft's MES solution provides the Track & Trace module for realizing ISA-95's goal for Inventory Operations Management. Refer to Track & Trace Overview for more details.

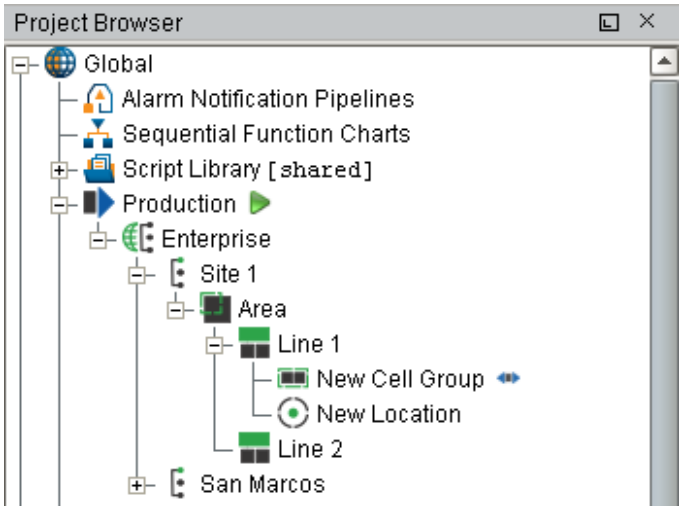
7 Production Model Overview

7.1 What Is The Production Model?

When any of the core MES modules (OEE, SPC, Recipe, T&T) are installed, the Production Model is added to the **Global** project resources in the Project Browser window of the Ignition Designer. The Production Model allows you to define your manufacturing process in a tree view form and provides an organized way to configure, control, and analyze your manufacturing activities. It provides the foundation on which the MES modules are built.

The Production Model is a hierarchy of Sites, Areas, Lines, Cell Groups, Cells, Locations, Storage Zones and Storage Units. Typically, Lines and Cells are used to represent machinery or equipment where a process occurs transforming raw materials into sub-assemblies or finished goods. Storage Zones and Storage Units are typically used to define where to get or store material.

Lines and Cells defined in the production model should be considered to be equipment that is bolted to the floor and has conduit running to it. Mobile equipment such as pallets, bins, dies used for pressing, etc. are not defined in the production model, but configured in the MES Management screen as Supplemental Equipment (Track & Trace only).



Below are the different types of Production Items that can be added to the production model.

Icon	Production Item	Description	Module
	Enterprise		All

Icon	Production Item	Description	Module
		The enterprise is the highest level of the production model and typically represents a manufacturing company. You can rename the Enterprise production item to your companies name. You can only have one Enterprise item in the Production Model.	
	Site	A site is a fixed geographical production location that is part of an enterprise. Separating your enterprise into multiple production sites allows for comparing OEE, downtime and production information between them.	All
	Area	An area is a physical or logical grouping of production lines.	All
	Line	A line is a collection of one or more cells and/or cell groups that work together to perform a sequence of process steps. Typically, the product flows from one cell or cell group to the next in sequence until the product, or sub assembly, being produced is complete. Understanding how Operations schedules or controls a production run will help in determining whether cells should be grouped into a line or be considered lines themselves.	All
	Location	A location item is the place where a sample is collected. This can be placed under an area or a line.	SPC
	Cell Group	A cell group contains two or more cells. Typically, these cells occur at the same time in the sequence of the line instead of one after another, causing the cell group to act as a single sub process or step within the production.	All
	Cell	The cell is a single machine, sub process or step required in the manufacture of a product. The product may be a hard product such as used in packaging, adding liquid or powder, etc. Packaging machines are a common example, but a cell applies to processes also.	All

Icon	Production Item	Description	Module
	Storage Zone	A storage area such as a warehouse.	T&T
	Storage Unit	A storage unit located inside of a storage zone. For example, you may have a warehouse with bay 1 to 5.	T&T

7.2 Configuring the Production Model

The production model is configured within the Ignition designer and is accessed by selecting the **Production** node under Global in the project browser. From here your enterprise, site, area (s), line(s) and line cell(s), line cell group(s), storage zone(s) and storage unit(s) can be added, renamed and deleted.

It is extremely important to understand production OPC values have an OPC item path that matches the layout of the production model and that renaming production items can cause Ignition tags associated with a production item to stop being updated.

7.2.1 Adding a New Production Item

To add a new Production item, right-click on the Production model and select the **New Production Item** > **New Production xxxx** menu item.

7.2.2 Renaming a Production Item

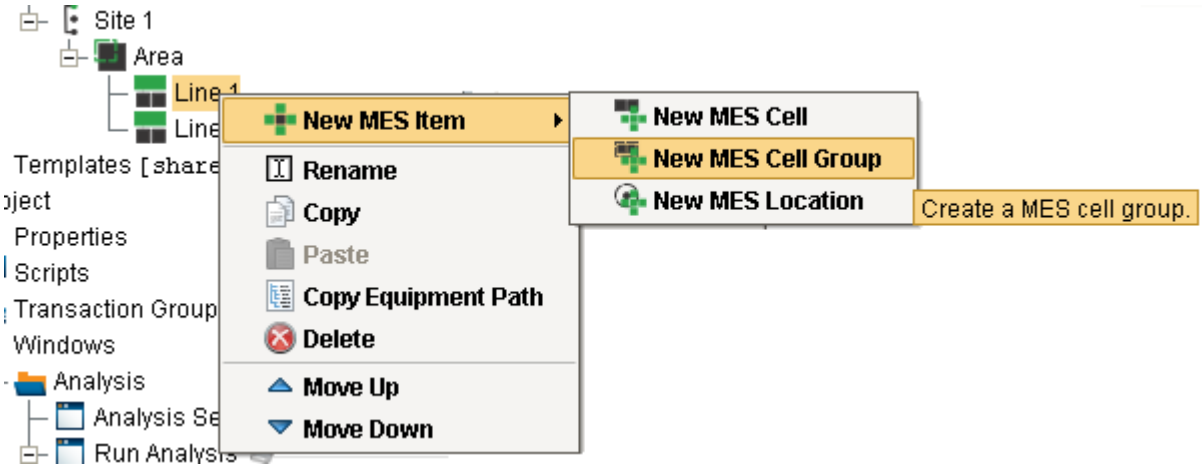
To rename a production item, right-click on it and select **Rename**, then enter the new name.

Please note that when you rename a production item, it actually creates a new instance of a production item and disables the old production item. This is important to note as data captured against that production item will not be accessible to the newly renamed production item. Spend the time to get the Production Item named correctly at the beginning of the project.

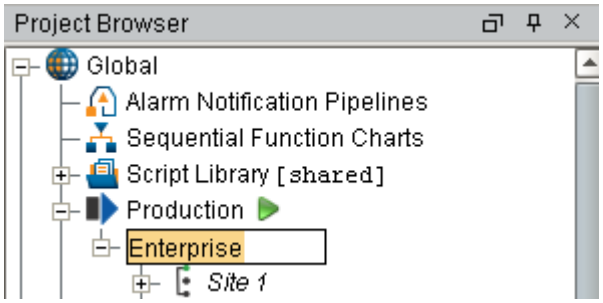
7.2.3 Deleting a Production Item

To remove an existing production item, right-click on the item and select the **Delete** menu item. A window will appear confirming that you permanently want to delete the production item.

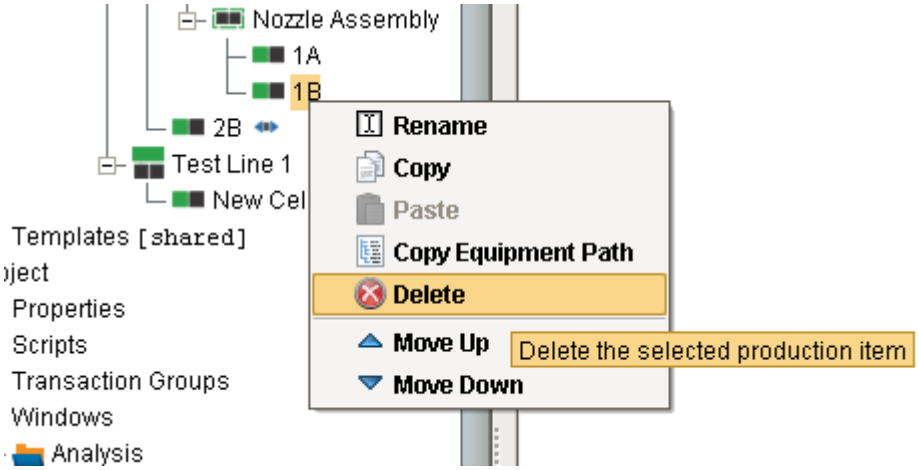
> Please note that any line(s), cell(s), cell group(s) and location(s) underneath the production item will also be permanently removed.



Adding a new Cell Group to the Production Model



Renaming the Enterprise



Delete a Cell

7.2.4 Copying a Production Item

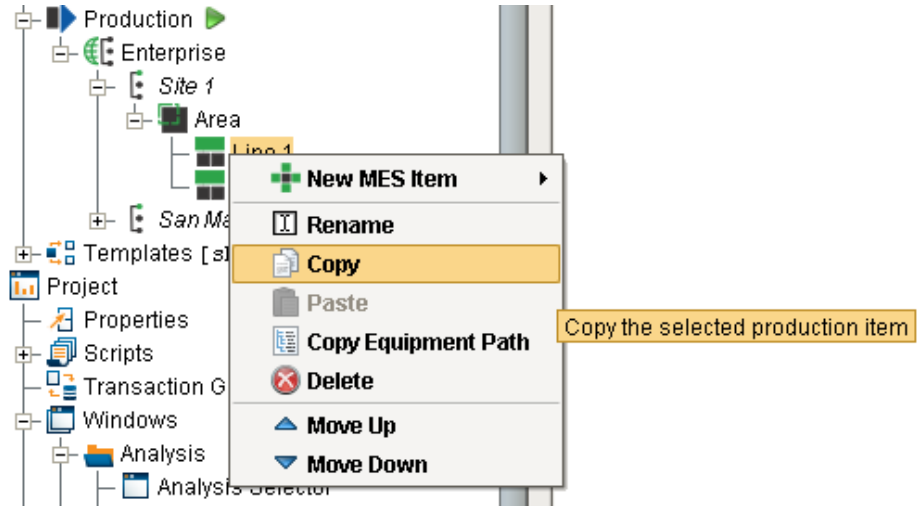
Right Click mouse button and select **Copy** on any production item to copy that production item.

Right Click mouse button and select **Paste** to make a copy of that production item in the production model.

If you are copying a line, select the line before copying it. When you paste it, select the area in which to the create a copy of that line.

Good Practice

It is recommended that you make a gateway backup prior to copying and pasting Production items. It is not recommended that you make changes to the production model on the production server without scheduling with Operations and having the system backed up.



Copying a Production Item

7.3 Production Item General Settings

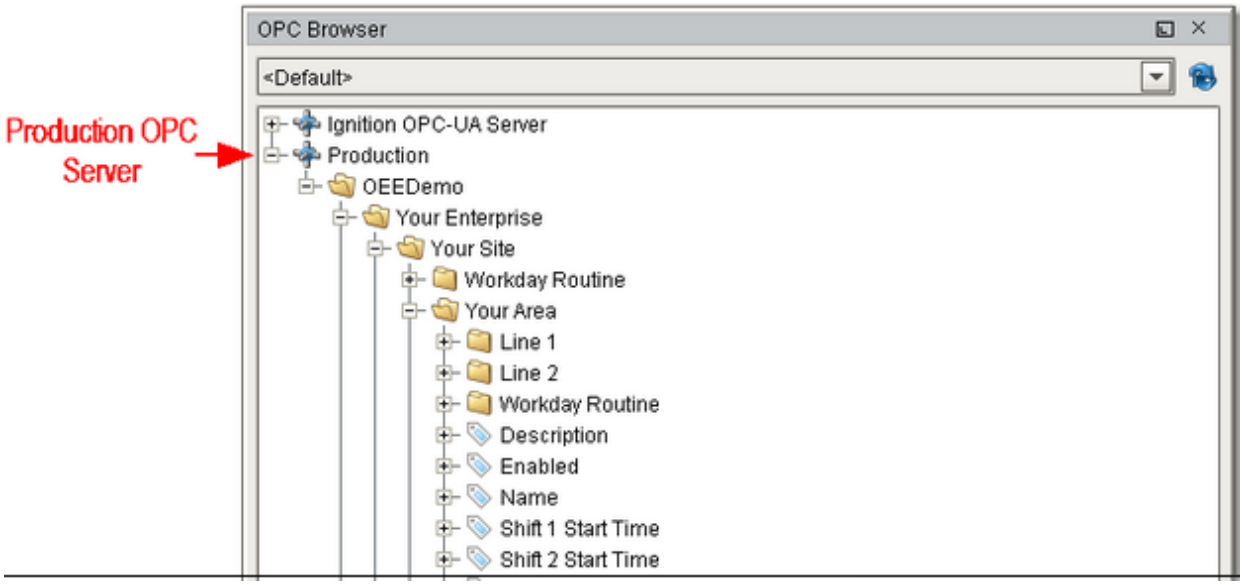
The general settings are accessed by selecting the desired production item and selecting the General tab.

Setting	Description
Enabled	By default, the production item is enabled. It can be disabled by un-checking the Enabled setting and saving the project. This will stop the track and trace, OEE, downtime, SPC, recipe and scheduling modules from using the area and any other production items that are underneath it.
Description	This is an optional description and is just for your reference.

7.4 OPC Production Tags

As production items are added to the production model, run time access into configuration settings and current state of those production items is available through the Production OPC Server. It is added automatically when MES Modules are installed. When the production items are added, removed or modified, the changes will be reflected in the Production OPC Server when the project is saved in the designer.

Please refer to the OPC Production Server Tag Reference in the Appendix for more help.

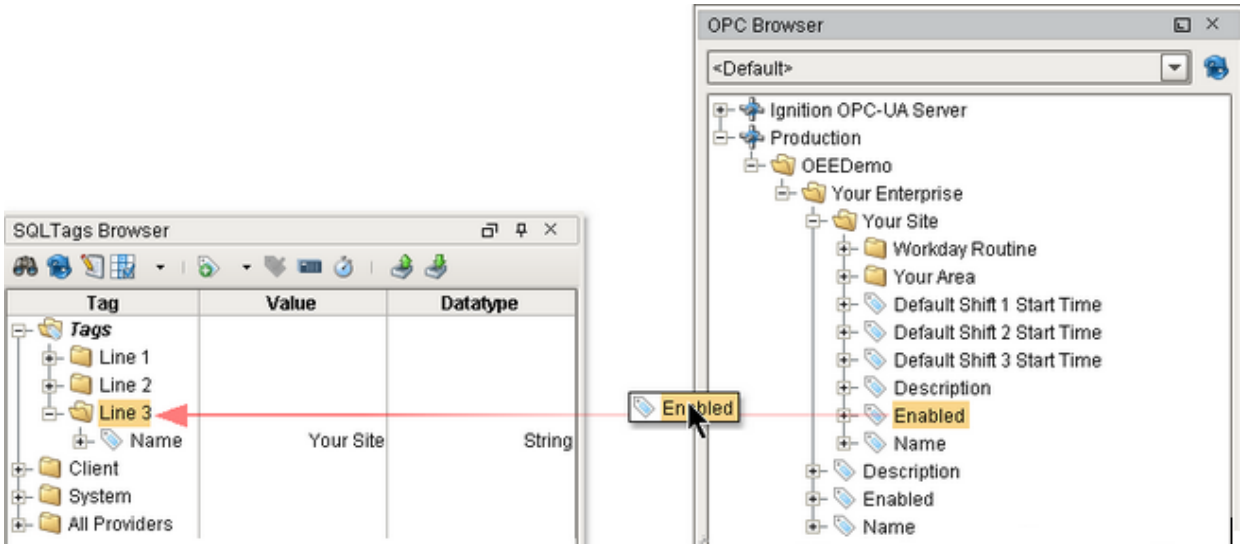


Demo OPC Values

7.4.1 Using OPC Production Tags in Your Project

Before Production OPC Server tags can be used in your project windows, transaction groups etc., they must be added to the Ignition SQLTags. This is done in the designer by selecting the SQLTag Browser and clicking on the OPC icon. This will cause the OPC Browser to appear. Next, drill down in the Production node within the OPC Browser. Drag the desired Production OPC Values over to the SQLTag Browser as shown.

> When writing to OPC values that are related to production model settings, the new value is not retained upon restarting. This is because production model settings are saved in the Ignition project and is only saved when done so in the designer.



Add Production OPC Server Values to SQLTags

7.5 Enterprise Settings

7.5.1 General Tab

These settings are accessed by selecting the enterprise item contained in the Production folder in the project browser and then selecting the **General** tab as shown.

By default, the enterprise production item is enabled. It can be disabled by un-checking the **Enabled** setting and saving the project. This will stop the MES modules from executing the enterprise and all other production items that are underneath it.

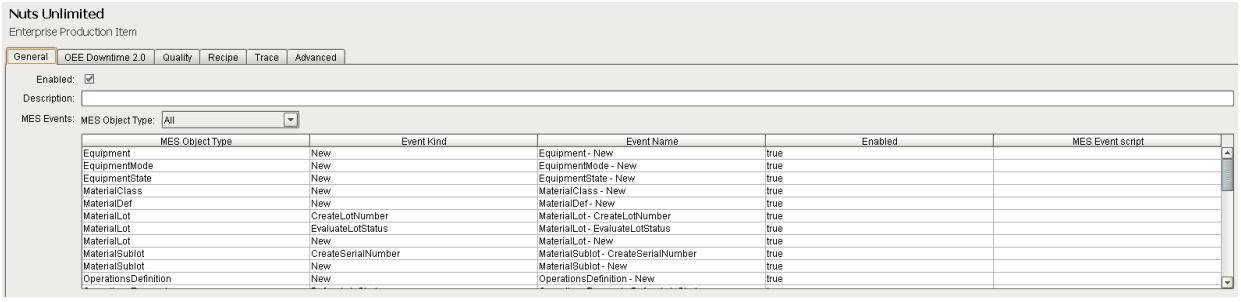
MES Events

MES Events are defined for the different types of MES objects. See MES Events in the MES Object section of the reference guide for more information. MES system events are generated by the Track & Trace and OEE 2.0 module.

There are two types of events, System and Custom.

System events are predefined and cannot be added, deleted or renamed, but do allow entering script to execute when the event is triggered.

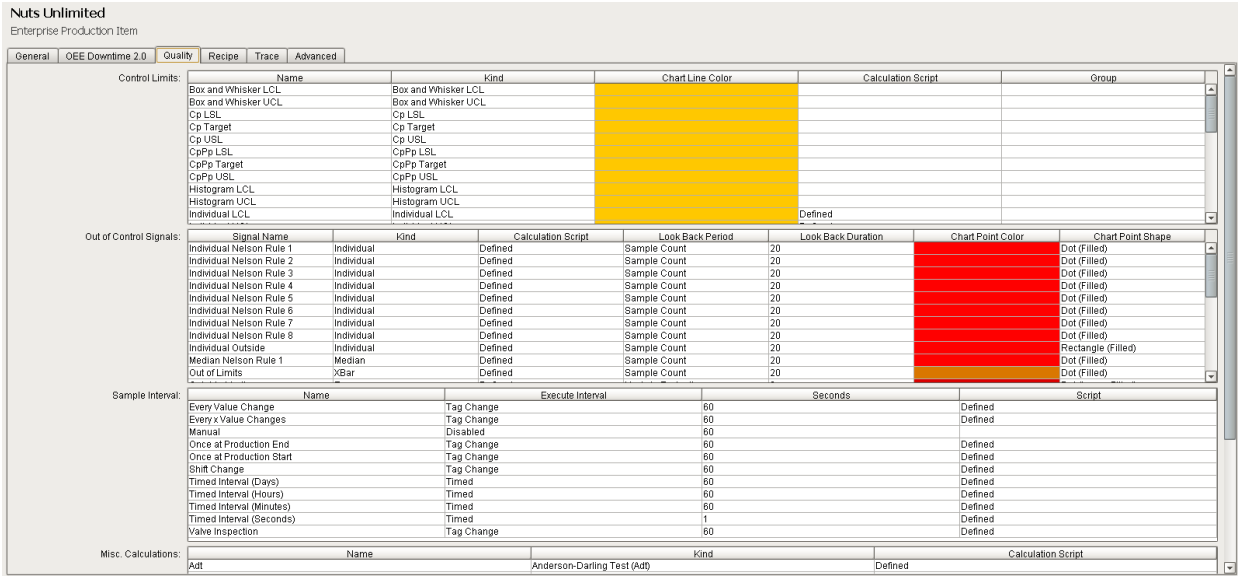
Custom events can be added, deleted or renamed and are executed from script.



7.5.2 Quality Tab

The Quality tab is visible when the SPC Module is installed and contains settings for Control limits, signals, sample intervals and misc. calculations.

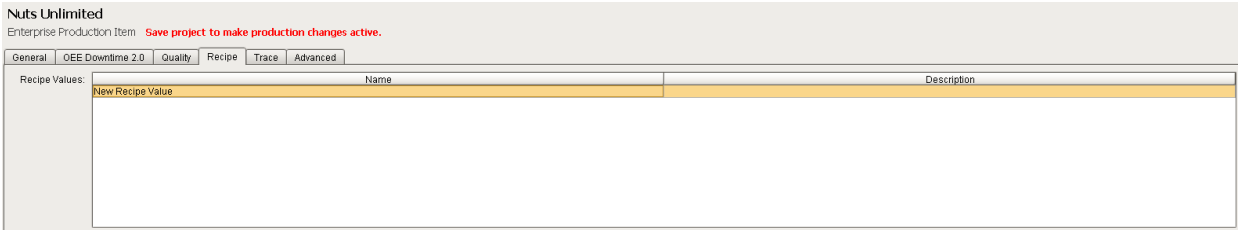
How Control limits, signals, intervals and misc. calculations are calculated are defined here and can be edited. For more information on SPC configuration, please refer to SPC Production Model Configuration section.



7.5.3 Recipe Tab

Recipe Values names can be defined at the enterprise level. Anything defined at the enterprise level will become available at the Site, Area, Line, Cell and Location level. This provides a way to manage recipe values that may be common across many lines at your site as every time you create a line, those values will be there. It also gives you the ability to assign a default value at that level that will carry down (be inherited) to subsequent levels.

For more information please refer to the Recipe Production Model Configuration section.



7.6 Site Settings

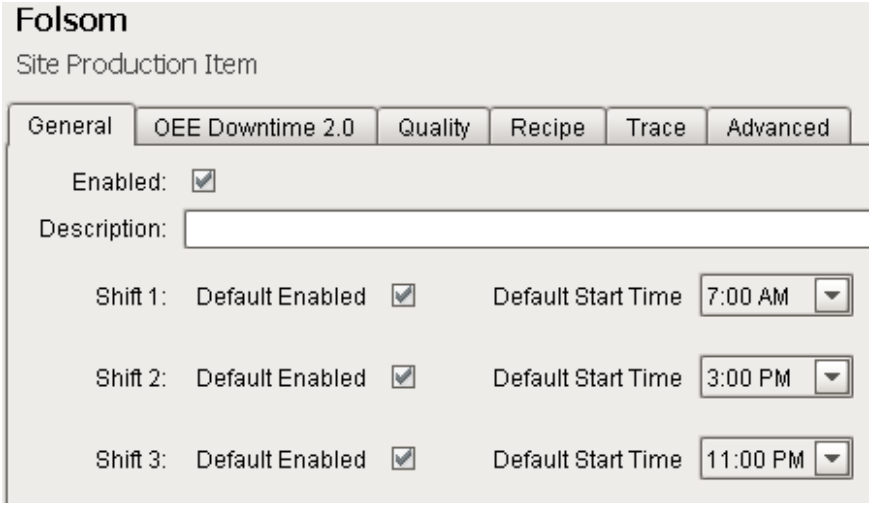
7.6.1 General Tab

Default shift times can be configured on this tab. You can define how many shifts exist, i.e. two 10 hour shifts or three 8 hour shifts.

The default start time for each shift is defined here. The shift end time is determined by the start time for the following shift.

If you only have one 8 hour shift, set the end time of that shift to be the start time of the next shift and uncheck **Default Enabled** for the next shift.

Areas and lines under this site can be configured to inherit from the default shifts for the site, or they can be overridden on the general tab for that area or line.



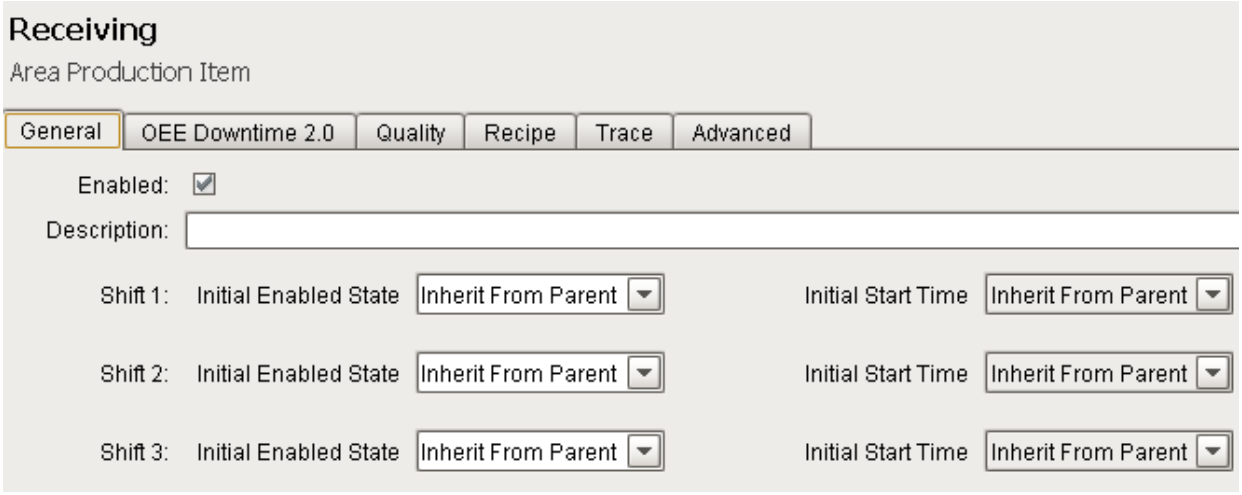
7.6.2 Recipe Tab

Recipe Values names can be inherited from the Enterprise level or defined at the Site level. For more information please refer to the Recipe Production Model Configuration section.

7.7 Area Settings

7.7.1 General Tab

Areas can be configured to inherit from the default shifts for the site, or they can be overridden in this tab.

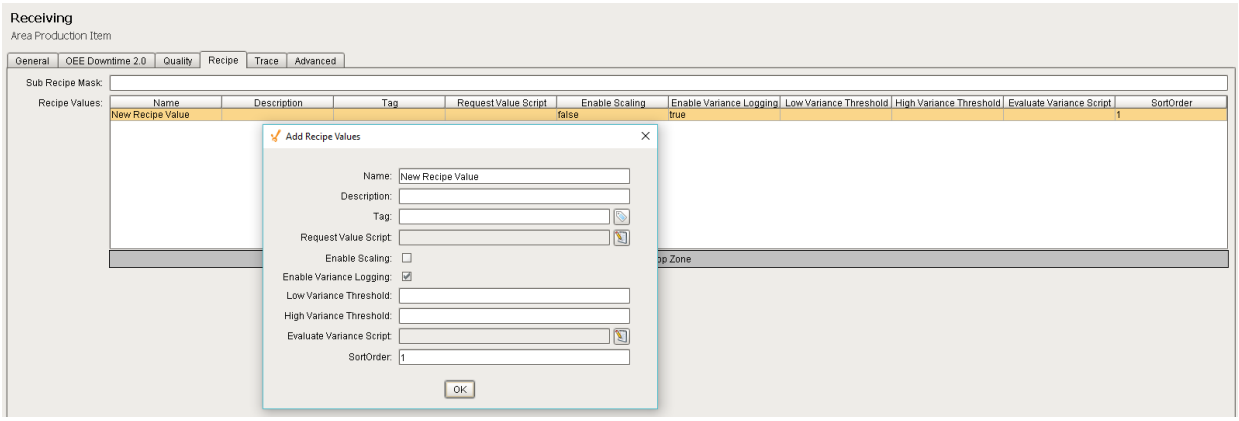


7.7.2 Recipe Tab

Recipe Values names can be inherited from the Enterprise or Site level settings, or defined at the Area level.

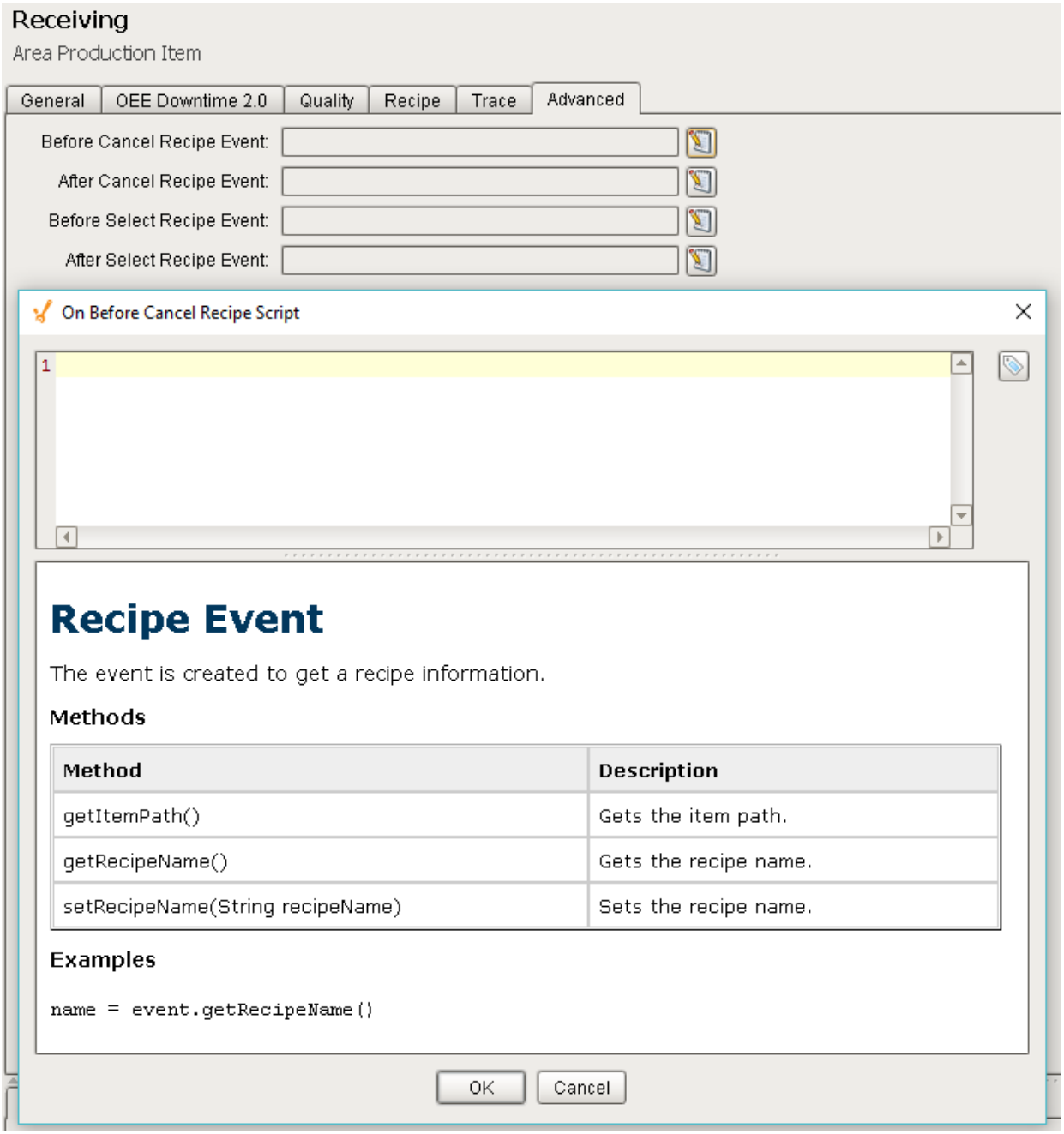
At this level, it is possible to add configuration information regarding the tag, variance thresholds and the scripts used to evaluate them.

For more information please refer to the Recipe Production Model Configuration section.



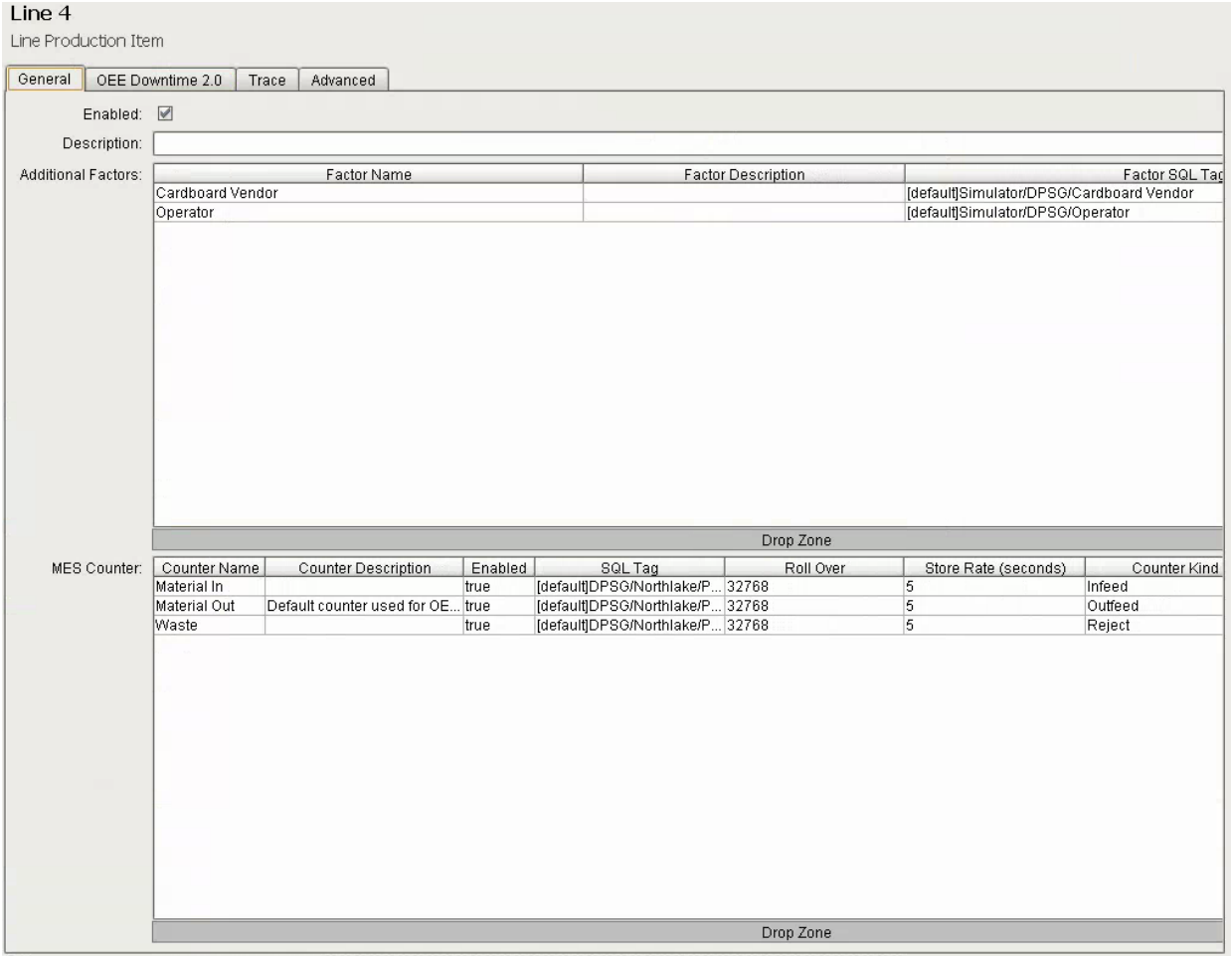
7.7.3 Advanced Tab

At the Area level, events generated by the Recipe Module are exposed and custom scripting regarding what happens when a recipe is selected or canceled can be defined.



7.8 Line Settings

The Line Production Item provides a configuration page for setting up a Production Line for all the MES modules. The General and Advanced Tab are always present whereas each module when installed provides an additional tab.



7.8.1 General Tab - Additional Factors

The OEE Module collects and logs a number of downtime and production data values. However, what if other values outside of downtime and production values are of interest? Additional factors are the solution. Additional Factors are user defined data points that are logged along with the production and downtime information. Once they are logged, they can be shown in charts, tables and reports. Additionally, other analysis can be done by filtering and/or setting up comparisons by their values.

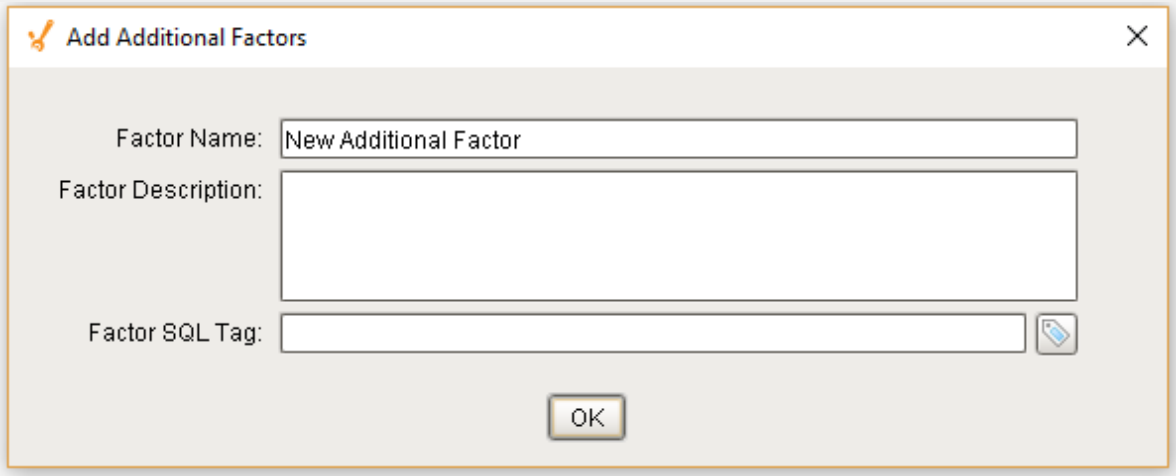
Additional Factors can added to the Line, Cell Group and Cell Production Items in the Production Model Designer.

Any value that can be read from an Ignition SQLTag can be added as an additional factor. This includes values derived from scripts, or from barcode readers, databases, calculations, PLCs, etc.

Any tag can be added as an additional factor. To configure, select a Line in the production model and select the **General** tab on the right. Right click on the **Additional Factors** table and select **New**. An additional factor is simply just a name and a tag.

Properties

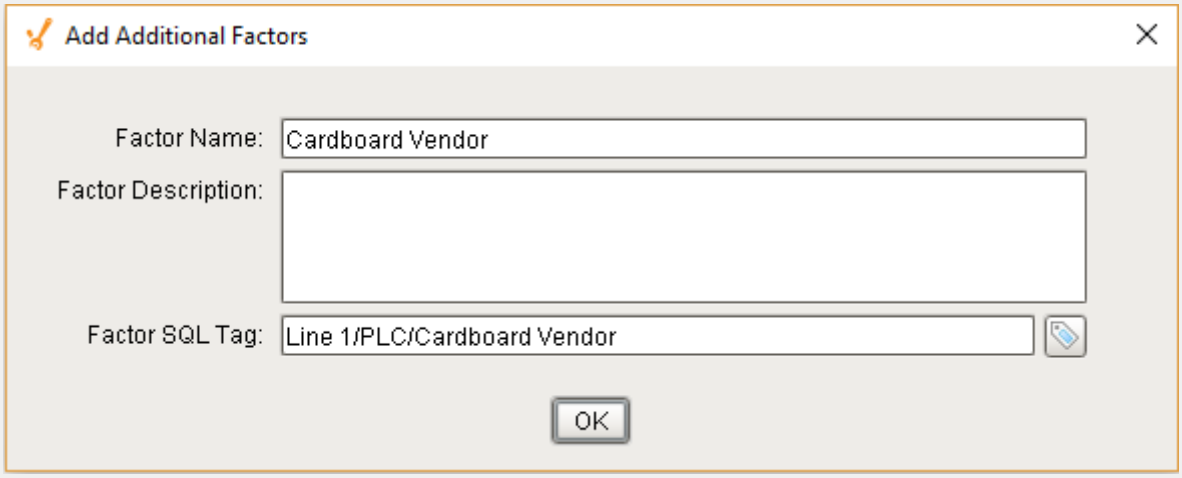
Factor Description	Optionally, this property can be set to a description for the additional factor. It is not used by the OEE Downtime and Scheduling Module other than for reference.	String
Factor Name	This reflects the name of additional factor that is configured in the designer.	String Read Only
Factor SQLTag	This reflects the Factor SQLTag setting that additional factor is configured for in the designer. It is the name of SQLTag to read the factor value from.	String Read Only

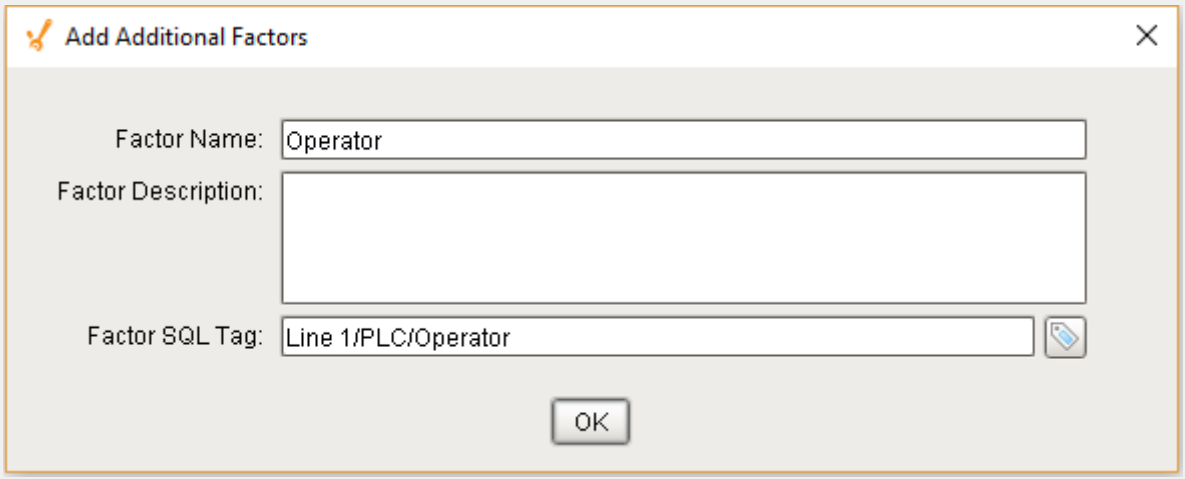


Example

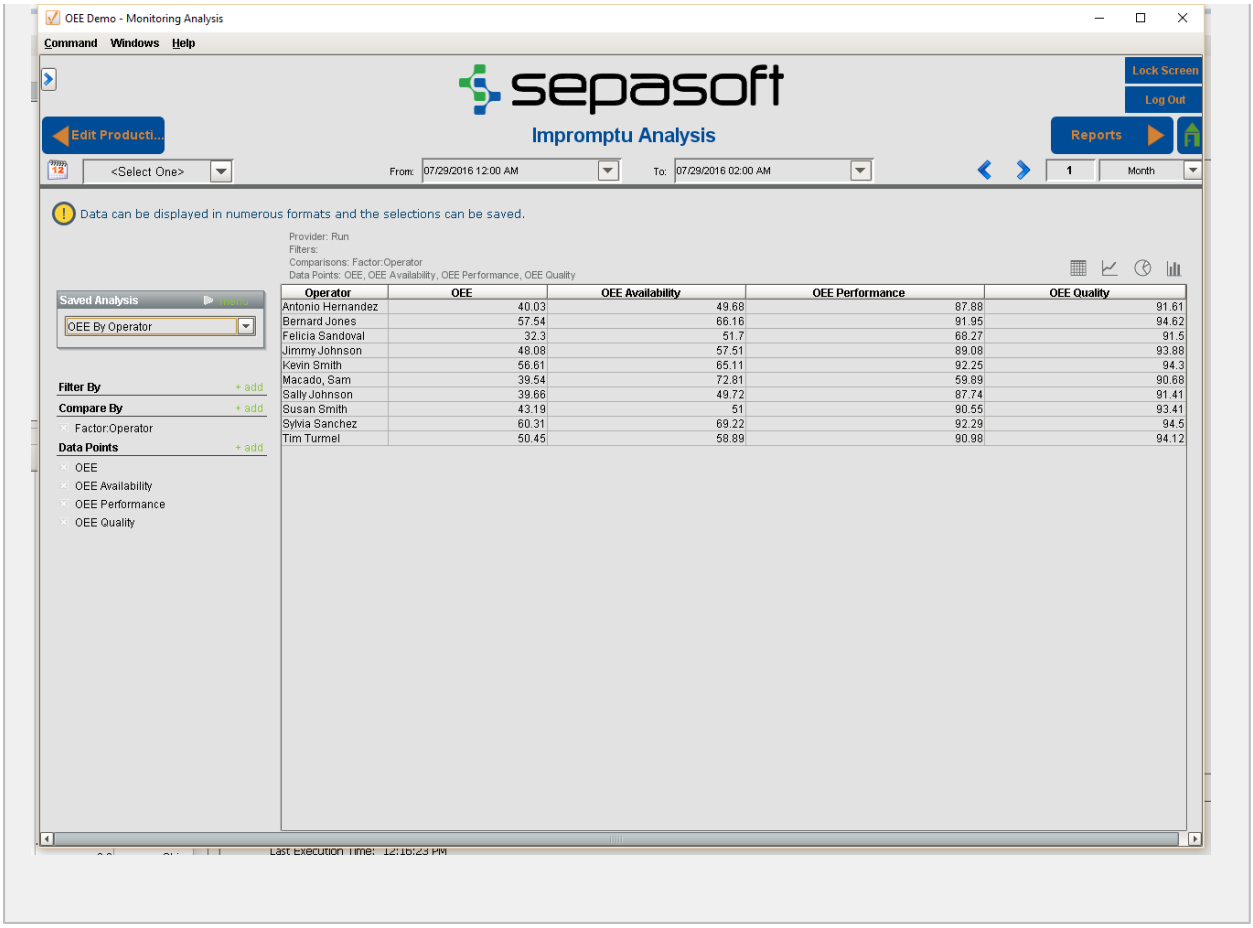
In the example, we have two factors, **Cardboard Vendor** and **Operator**. The operator can select the vendor that provided the cardboard or it can be obtained from some other source. Now, OEE and downtime results can be shown for each cardboard manufacturer. This can identify quality problems with raw material that directly affect production efficiency. With the operator setup as an additional factor, the operator's name will be

logged along with the production and downtime data. By doing so, OEE and downtime information can be filtered and grouped by the operator name. But this could just as well be the production crew, supervisor, maintenance crew or any other user defined value that can be monitored or entered into the system.





Adding these factors to a production line will allow us to capture the value of these tags whenever they change. In the impromptu analysis, we can then compare OEE values by our additional factor: Operator



7.8.2 General Tab - MES Counters

The MES Counters are used to associate Process Segments (Operations) with production counts. MES Counters record production counts 7/24, independent of scheduled production runs.

Counter names and the associated tag are defined in the Production Model. In the MES Management screen, the Quantity Source of Infeed and Material Process Segments can be set to use these MES counters.

MES counters are available for the Line, Cell Group, Cell or a Storage Unit production items in the Production Model Designer.

It is recommended to set up MES Counters at the cell level in order to accurately capture counts while indexing product on the line. Addition configuration must be accomplished with the cell settings in the production model.

The quantity from the MES counters can be obtained through scripting, see system.mes.getCountValue.

Adding MES Counter

To configure an MES Counter, select a Line, Cell Group, Cell or a Storage Unit in the production model and select the **General** tab on the right. Right click on the **MES Counter** table and select **New**. Properties can be set through the **Add MES Counter** Window.

Counter Name

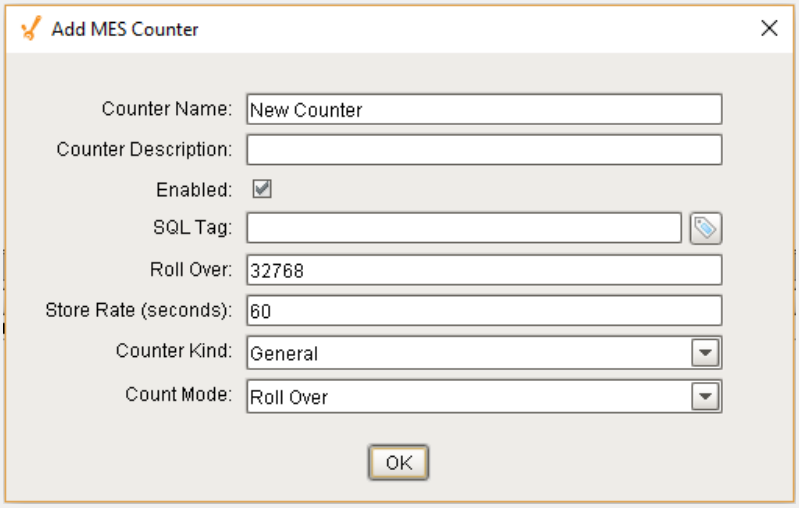
Name of the counter.

Counter Description

The description for the MES counter. This setting is not mandatory.

Enabled

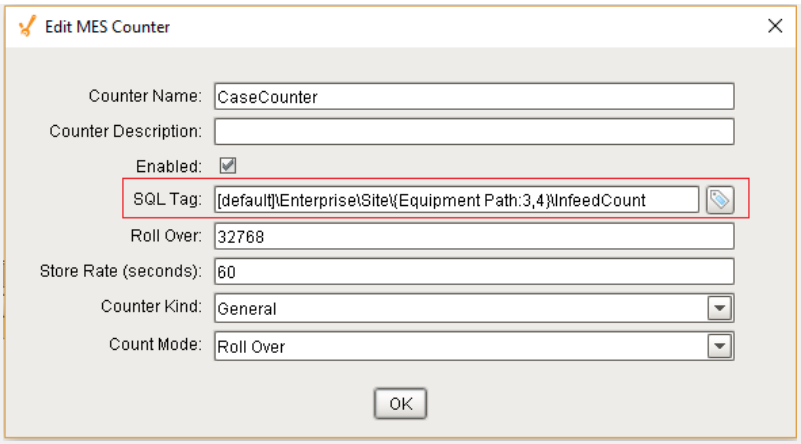
The counter can be enabled or disabled here.



SQL Tag

The path to the Tag Provider and ignition tag where the count value will come from is assigned to the MES counter here.

Parameterized Tag Paths can be used here which allows for indirection and exported MES Counters to be easily deployed to other equipment.



Roll Over

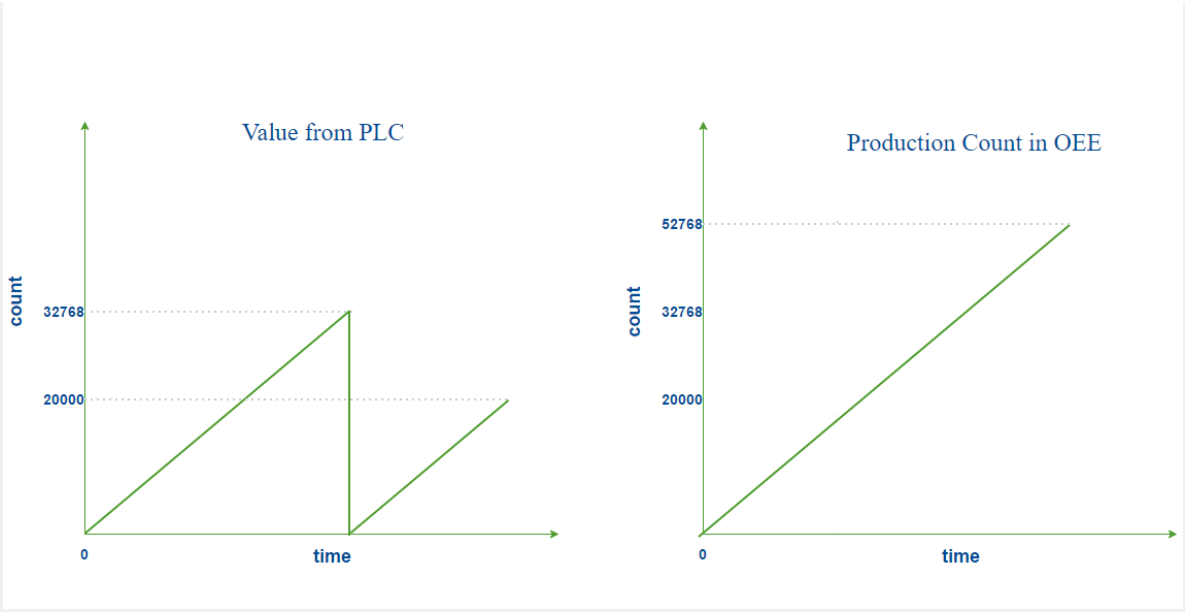
For PLC count tags that that do not get reset, they will eventually reach a finite maximum value at which point, the value will 'rollover' back to zero. The Roll Over setting allows you to define the value that should be added to the count tag whenever a roll over occurs. By default it is 32768 which equates to a 16 bit signed data value. Your setting will be dependent upon the datatype of your plc count tag.

During a production run, the incoming count value is added to the Roll Over setting multiplied by the number of times a rollover has occurred.

Example: production count value = incoming count value + 32768 * 3

Production Count	PLC Count tag	Rollover	Calculation
32700	32700	0	32700 + 32768 * 0
32750	32750	0	32700 + 32768 * 0
32918	150	1	150 + 32768 * 1
33068	300	1	300 + 32768 * 1

This value is only used when the **Count Mode** is set to **Rollover**.



Roll Over Count Mode

Store Rate

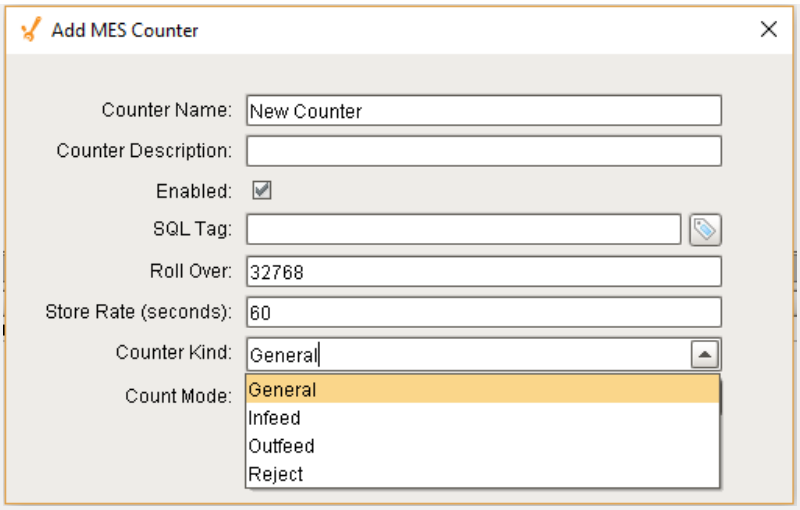
The MES counter will be captured and stored in the database after this specific interval in seconds if the value has changed If **Store Rate** is set to zero, every value change will be recorded.

Counter Kind

MES Counters can be set to four different kinds:

- Infeed
- Outfeed
- Reject
- General

Infeed, **Outfeed** and **Reject** kinds are used solely by the OEE module to determine which MES counter to use for OEE Performance, OEE Quality and production count information. The **General** kind can be used for any other count value.



Count Mode

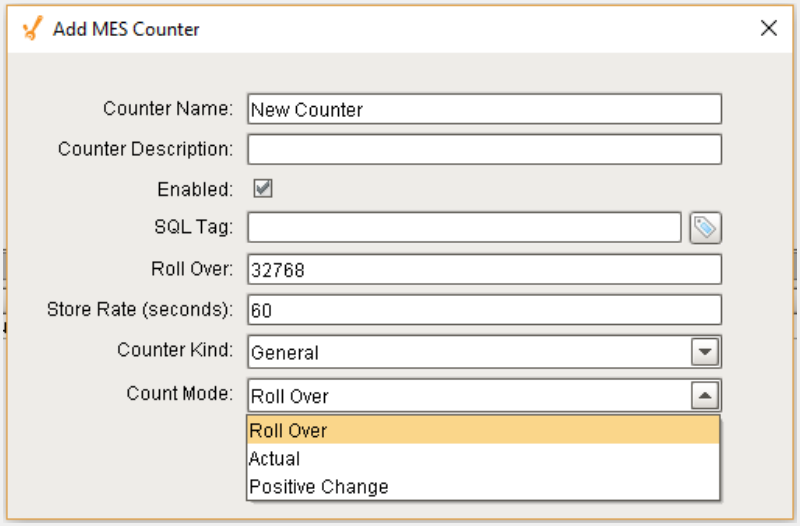
The Count Mode can be set to **Roll Over**, **Actual** and **Positive Change**.

Roll Over

See Rollover section for this count mode.

Actual

The Actual count mode simply uses whatever value is passed through the sql tag to represent the actual production counts. Production counts can go down as well as up.

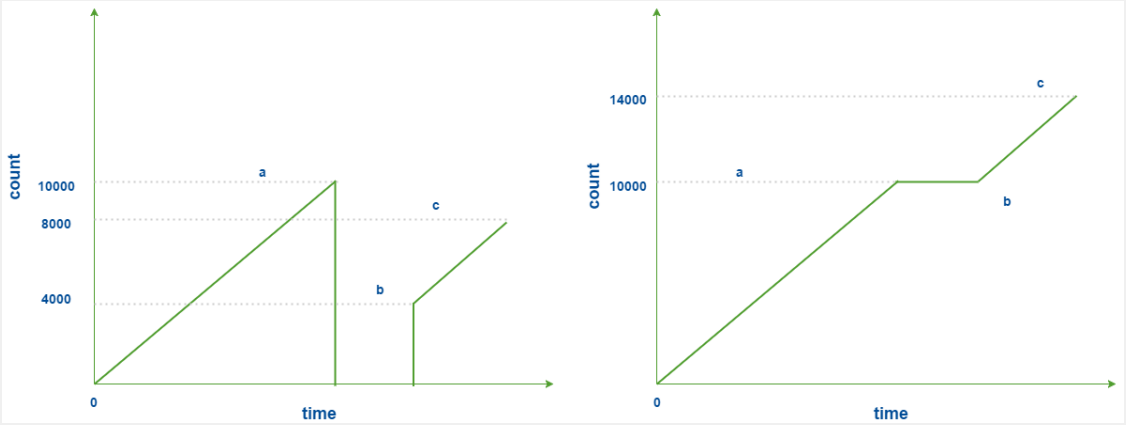


Positive Change

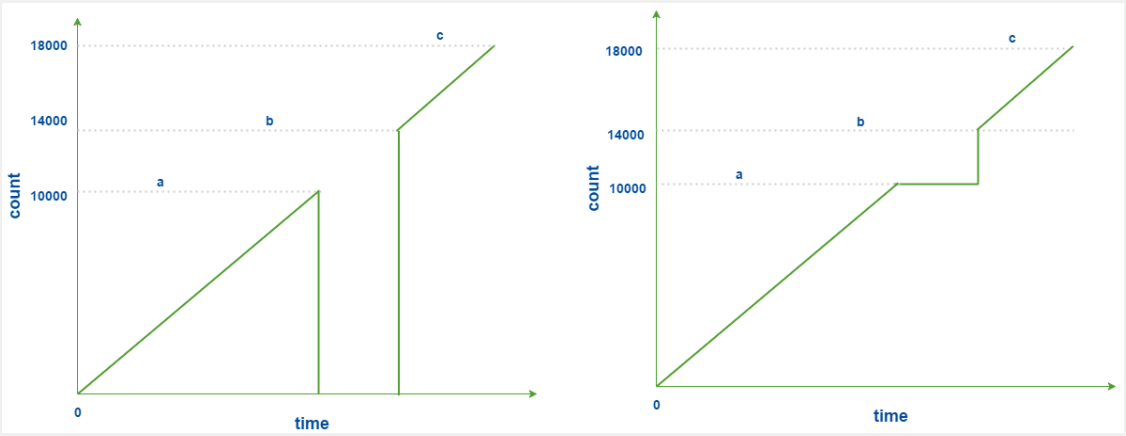
The Positive Change mode ignores any sql tag count values that are zero and will accumulate the counts. Three different cases are illustrated using the graphs shown.

PLC Count vs. OEE Count

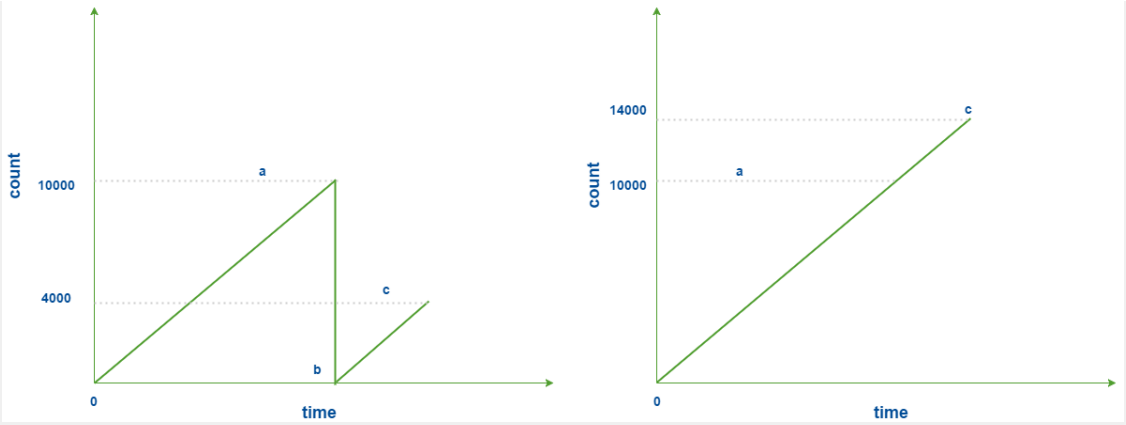
Case 1



Case 2



Case 3



Counter Rapid Development Features

Not only do counters allow for parameterization (as shown above), they also support copy, paste, import, and export features for rapid development. Configure the infeed counter for one Production Model Node (i.e. a Cell or Line) with parameterization, then copy and paste it to the other cells. Alternatively, copy and paste the Node itself and rename it for a similar effect.

7.8.3 OEE 2.0 Downtime Tab

The OEE Downtime 2.0 tab is specific to the OEE module and is available for the Line, Cell Group and Cell Production Items. A number of configuration settings are provided that can be used to obtain equipment mode, state and count values from ignition tags (whether plc tags, memory or expression tags).

Downtime Detection Mode

How line downtime is determined can be changed based on the selected Downtime Detection Mode. Valid options for the downtime detection mode are...

- Equipment State
- Key Reason (Cell Priority)
- Key reason (Neighbor Priority)
- Initial Cell

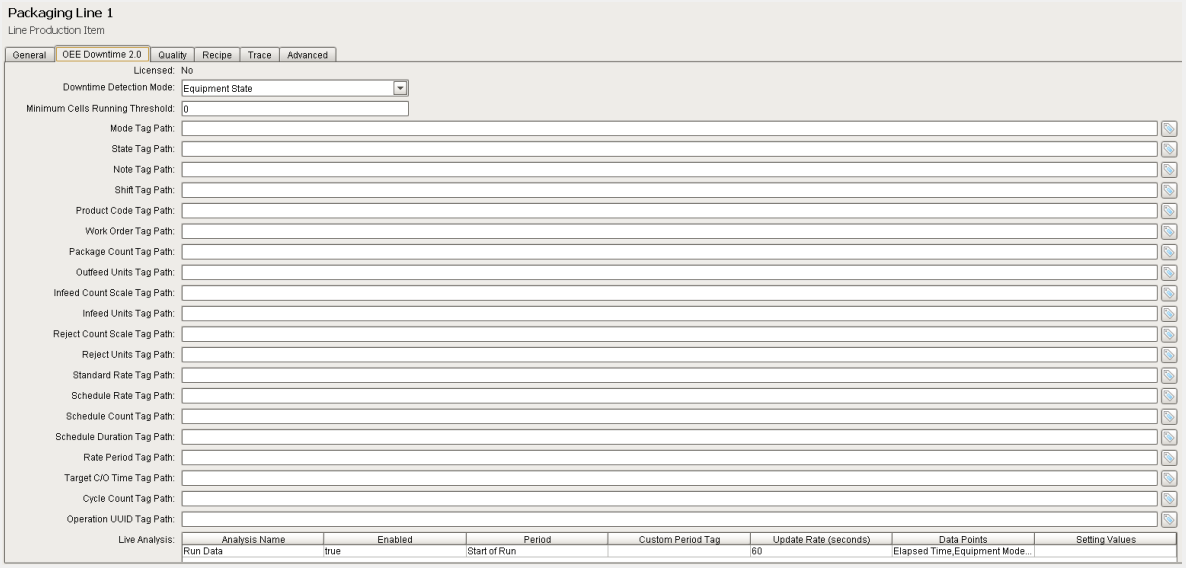
- Parallel Cells

Refer to Downtime Detection Mode for more information on the various Downtime Detection Methods.

> ⚠ Downtime Detection Mode is only available for the Line and Cell Group Production Item.

Minimum Cells Running Threshold

Minimum Cells Running Threshold determines how many cells in the Line (or Cell Group) must be running in order for the Line (or Cell Group) to be considered as Running.



Tag Collector Paths

Tag Collectors are provided to allow any of the parameters needed to drive OEE Metrics to be provided externally to the OEE module. Virtually all the Tag Collectors can be left blank in which case the OEE engine will determine the value from product code configuration information as defined in the OEE Material Manager or from internal calculations. Exceptions to this would be the equipment state and counts where needed.

> ⚠ When adding a tag to a Tag Collector, you must use memory tags. The Production Model will write values to any tags defined here as well as read the value whenever it changes from an external source.

Tag Collector	Tag Type	Data Type	Description

Mode	Memory	Integer	The Mode Tag, if provided, will be written to by the Production Model whenever the equipment mode changes based on Material Production Settings. The value of the mode tag can also be written to whenever the Mode value changes either indirectly from a plc tag (via tag change event) or from the HMI. The value of the Mode tag will be recorded for the current mode.
State	Memory	Integer	The State Tag path will generally come from a plc as the source of the current equipment state. Exceptions to this are at the Line level when using a downtime detection method other than Equipment State.
Downtime Note	Memory	String	Apart from scripting and Downtime table component, the downtime notes can be added by using tags.
Shift	Memory	String	When left blank, shifts defined in the Ignition Schedule Management component and defined in the Equipment Manager for a line will be used to determine the current shift. If a tag is provided here, whatever value is in the tag e.g. 'Shift A' will be recorded for the current shift.
Product Code	Memory	String	When left blank, the Product Code currently running on the line will be determined from the scheduled run as selected by the Scheduler or Run Director component. When a tag path is provided, the product code for the line or equipment (cell) will be determined from the value of the tag.
Work Order	Memory	String	When left blank, the Work Order currently running on the line will be determined from the scheduled run as selected by the Scheduler or Run Director component. When a tag path is provided, the product code for the line or equipment (cell) will be determined from the value of the tag.

Tag Collector Path	Tag Type	Data Type	Description
Package Count	Memory	Float	When left blank, the Package Count will be determined from the Package count setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Package Count for the line or equipment (cell) will be determined from the value of the tag. For more information on the Package Count, refer to the Material Production Settings section.
Line Outfeed Units	Memory	String	When left blank, the Line Outfeed Units will be determined from the Line Outfeed Units setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Line Outfeed Units for the line or equipment (cell) will be determined from the value of the tag. For more information on the Line Outfeed Units, refer to the Material Production Settings section.
Infeed Count Scale	Memory	Float	When left blank, the Infeed Count Scale will be determined from the Infeed Count Scale setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Infeed Count Scale for the line or equipment (cell) will be determined from the value of the tag. For more information on the Infeed Count Scale, refer to the Material Production Settings section.
Line Infeed Units	Memory	String	When left blank, the Line Infeed Count Scale will be determined from the Infeed Count Scale setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Infeed Count Scale for the line or equipment (cell) will be determined from the value of the tag. For more information on the Line Infeed Units, refer to the Material Production Settings section.

Tag Collector Path	Tag Type	Data Type	Description
Reject Count Scale	Memory	Float	When left blank, the Reject Count Scale will be determined from the Reject Count Scale setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Reject Count Scale for the line or equipment (cell) will be determined from the value of the tag. For more information on the Reject Count Scale, refer to the Material Production Settings section.
Line Reject Units	Memory	String	When left blank, the Line Reject Units will be determined from the Line Reject Units setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Line Reject Units for the line or equipment (cell) will be determined from the value of the tag. For more information on the Line Reject Units, refer to the Material Production Settings section.
Standard Rate	Memory	Float	When left blank, the Standard Rate will be determined from the Standard Rate setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Standard Rate for the line or equipment (cell) will be determined from the value of the tag. For more information on the Standard Rate, refer to the Material Production Settings section.
Schedule Rate	Memory	Float	When left blank, the Schedule Rate will be determined from the Schedule Rate setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Schedule Rate for the line will be determined from the value of the tag. For more information on the Schedule Rate, refer to the Material Production Settings section.

Tag Collector Path	Tag Type	Data Type	Description
			Schedule Rate Tag Path is only available for the Line Production Item.
Schedule Count	Memory	Integer	When left blank, the Schedule Count will be determined from the scheduled run as selected by the Scheduler or Run Director component. When a tag path is provided, the Schedule Count for the line or equipment (cell) will be determined from the value of the tag. The Schedule Count provides the number of units scheduled to be produced. ⚠ Schedule Count Tag Path is only available for the Line Production Item.
Schedule Duration	Memory	Integer	When left blank, the Schedule Duration will be determined from the scheduled run as selected by the Scheduler or Run Director component. When a tag path is provided, the Schedule Duration for the line or equipment (cell) will be determined from the value of the tag. The Schedule Duration provides the expected runtime required for the number of units scheduled to be produced and is calculated by the Schedule Rate.
Rate Period	Memory	String	When left blank, the Rate Period will be determined from the Rate Period setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Rate Period for the line or equipment (cell) will be determined from the value of the tag. For more information on the Rate Period, refer to the Material Production Settings section.
Target C /O Time	Memory	Integer	

Tag Collector Path	Tag Type	Data Type	Description
			When left blank, the Target C/O (Changeover) Time will be determined from the Changeover settings for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Target C/O (Changeover) Time for the line or equipment (cell) will be determined from the value of the tag. For more information on the Target C/O (Changeover) Time, refer to the Material Production Settings section. ⚠ Target C/O Time Tag Path is only available for the Line Production Item.
Cycle Count	Memory	Float	When left blank, the Cycle Count will be determined by the OEE Module. When a tag path is provided, the Cycle Count for the line or equipment (cell) will be determined from the value of the tag.
Operation UUID	Memory	String	When left blank, the Operation UUID will be determined from the currently running Operation on the Line or equipment (cell). When a tag path is provided, the Operation UUID for the line or equipment (cell) can be determined from the value of the tag. The purpose for this tag is to be able to provide OEE analysis data when production runs are not scheduled or started using the Run Director or Schedule Selector components, or scripting functions. In this case a tag can be used to provide a Run Identifier value i.e. Run_4253_XX. The Analysis Selector provides the ability to pull the Operation UUID as part of analysis, whether it is an internally generated Operation UUID or a passed Run Identifier.

Tag Collector Paths can be parameterized with {Equipment Path} to utilize indirection and more rapidly implement the production model. See Parameterized Tag Paths for more details.

Live Analysis

Live Analysis provides a flexible way of customizing your application to provide a set of real-time tag values that can be accessed from the Ignition designer and used in your application to provide real-time production monitoring. Live Analysis is configured in the OEE 2.0 Downtime tab of the Production Model Designer for the Line, Cell Group and Cell production items. When a Live Analysis is created, a corresponding set of tags is created in the MES Tag Provider that provide the real-time status of those datapoints based upon the Period defined for the Live Analysis. You can create multiple Live Analysis and use those tags to drive HMI displays.

To create a new Live Analysis:

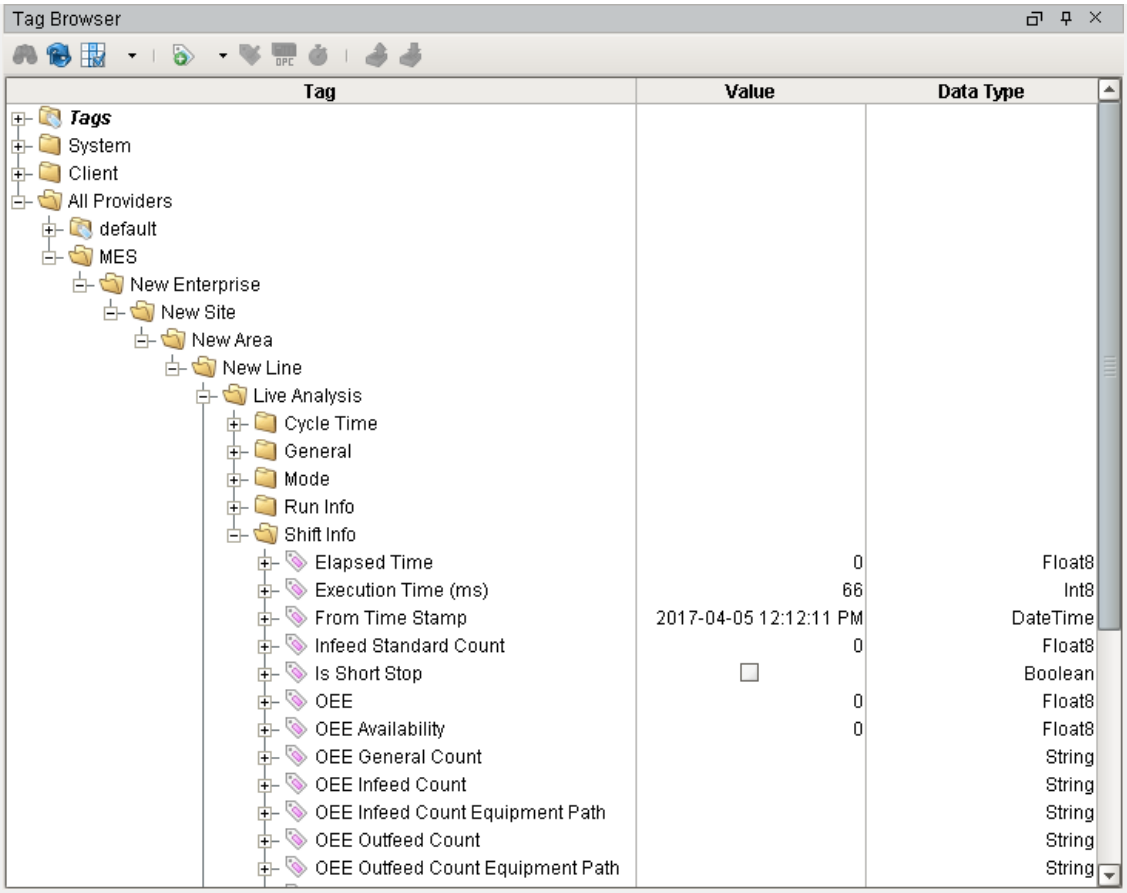
- Right click on the Live Analysis panel on the OEE 2.0 Downtime Tab in the Production Model Designer.
- Provide a Name
- Select the Period that the Live Analysis datapoints will return a value for. Valid options are Shift, Day (Midnight), Day (Production), Start of Run, Top of Hour, Custom Period Tag
- Select the frequency for how often the tag values will be updated. Default value is 60 seconds. Minimum value is 10 seconds
- Select the desired Data Points
- Add any further Settings Values required

⚠ You cannot select all Data Points in one Live Analysis. The maximum length string for Data Points is 1024 characters

Live Analysis:

Analysis Name	Enabled	Period	Custom Period Tag	Update Rate (seconds)	Data Points	Setting Values
Cycle Time	true	Shift		60	Average Normal Cycl...	
General	true	Shift		60	Delta Time Stamp,Eq...	
Mode	true	Shift		60	Equipment Mode Na...	
Run Info	true	Start of Run		60	Equipment Cell Orde...	
Shift Info	true	Shift		60	Elapsed Time,Infeed ...	

Live Analysis Settings Panel in the OEE 2.0 Downtime Tab



MES Tag Provider Live Analysis Tags

Live Analysis Settings

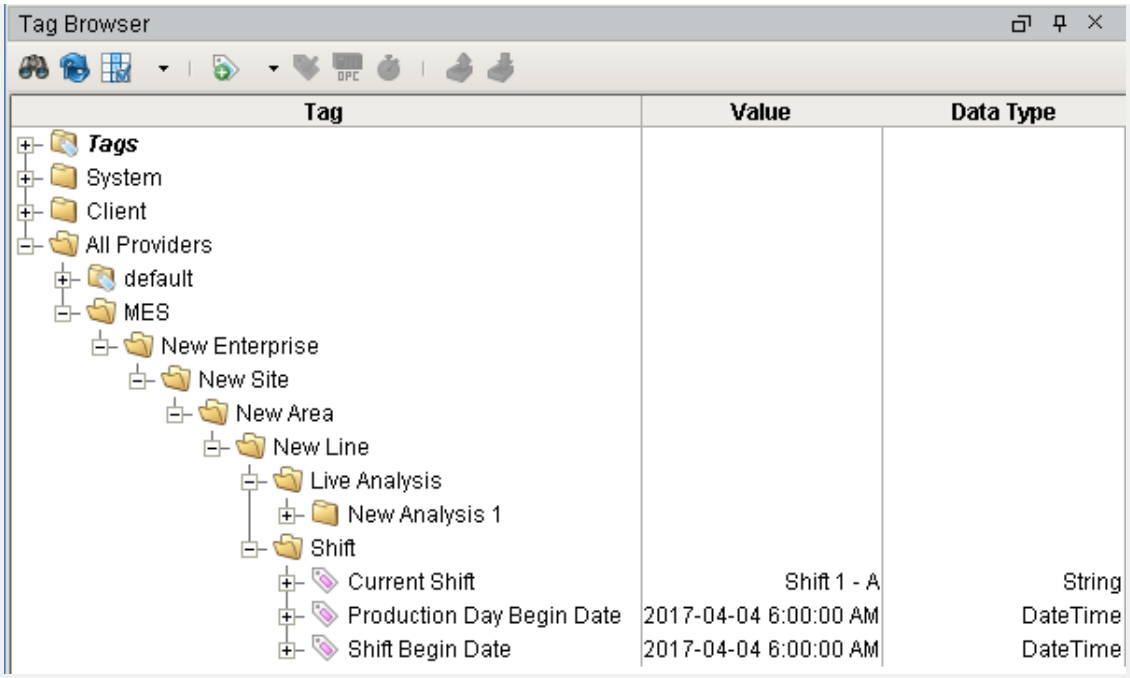
Setting	Description
Analysis Name	The name for the live analysis.
Enabled	The live analysis can be enabled or disabled with this setting.
Period	The duration of analysis can be set by **Shift**, **Day (midnight)**, **Day (production)**, **Start of Run**, **Top of Hour** or **Custom Period Tag**.
Custom Period Tag	A tag can be assigned to define the start datetime for a custom period. The end time will be the current time. It takes value in the date time data type. Example for a valid value for the custom period tag is: **`2017/04/04 14:00:00`**

Setting	Description
Update Rate	The rate in seconds by which the live analysis is updated. The minimum update rate is 60 seconds.
Data Points	Data points allows you to pick and choose the values you wish to access through tags. See the table below for the listing of available data points.

Shift Data Points

When creating a Live Analysis, the following shift data points will be automatically created.

Data Point	Data Type	Description
Current Shift	String	The currently running shift as defined in the Ignition Schedule Management component or passed from the Shift Tag Collector path
Production Day Begin Date	DateTime	Production start time
Shift Begin Date	DateTime	Start time of the current shift



Analysis Data Points and Settings are used by Live Analysis, the MES Analysis Selector and MES Analysis Controller components, the MES Analysis Data Source for reporting and the MES Analysis Settings object.

Equipment Data Points

Data Point	Data Type	Description
Equipment		
Equipment Cell Order	Int4	Integer value that determines the cell order of the equipment within the line. Is set to *null* for the line. Is set to 0 for first cell within each cell group
Equipment Name	String	Name of the equipment as defined in the production model
Equipment Note	String	Any note that has been recorded for this piece of equipment through the Note tag collector path in the Production model will be exposed here.
	DateTime	

Data Point	Data Type	Description
Equipment Operation Begin		Start Date time of the currently running operation on this equipment
Equipment Operation Sequence	Int4	The ordinal number (integer) of operation
Equipment Path	String	Production model path for this equipment
Equipment Type	String	Can be Line, Cell Group or Cell
Execution Time (ms)	Int8	Time taken to execute and update the Live Analysis. Used mainly for performance debugging
From Time Stamp	DateTime	Start Date Time of current data point results
Infeed Units	String	See Infeed Units for more details
Is Key Cell	Boolean	See Key Reason for more details
Operation UUID	String	Unique Identifier for currently running operation
Outfeed Units	String	See Outfeed Units for more details
Product Code	String	Product code currently being processed on this equipment
Rate Period	String	See Rate Period for more details

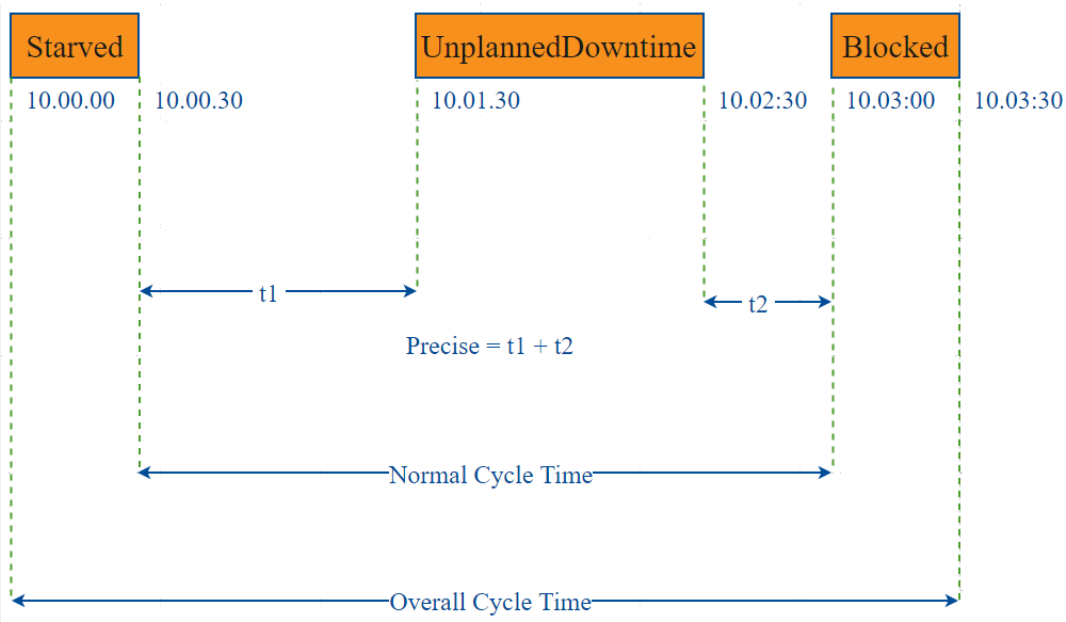
Data Point	Data Type	Description
Reject Units	String	See Reject Units for more details
To Time Stamp	DateTime	End Date Time of current data point results
Work Order	String	Work order currently being processed on this equipment

Equipment Count Data Points

Data Point	Data Type	Description
Equipment\Count		*Any defined counters for the production item will also appear in this folder
Equipment Infeed Scale	Float8	See Infeed Count Scale for more details
Equipment Package Count	Float8	See Package Count for more details
Equipment Reject Scale	Float8	See Reject Count Scale for more details
Outfeed-Material Out	String	Value of the default MES Counter used for OEE outfeed count

Equipment Cycle Time Data Points

The Cycle Time data points provides a number of metrics that can be used to measure the amount of time required to produce one piece. It is often used to gain an understanding of variations in production. Live Analysis provides Target, Normal, Overall and Precise Cycle Time metrics.



Data Point	Data Type	Description
Equipment\Cycle Time		
Relative Cycle Count	String	Relative Cycle Count is how many occurred for the compare by.
Target Cycle Time	Float8	Also known as Takt time, it is how often a piece must be produced to meet customer demand. It is often used to pace a production line, and it is a calculated number in seconds.
Total Cycle Count	String	Total Cycle Count is accumulative, it is sum total of all the cycle count.
Equipment\Cycle Time\Normal		**Normal Cycle Time is the actual cycle ignoring the equipment states like starved, blocked, etc.**
Average Normal Cycle Time	Float8	Average Normal cycle time in seconds for the time period selected
	Float8	

Data Point	Data Type	Description
Max Normal Cycle Time		Max Normal cycle time in seconds for the time period selected
Min Normal Cycle Time	Float8	Min Normal cycle time in seconds for the time period selected
Normal Cycle Time	Float8	Normal Cycle Time in seconds is the actual cycle ignoring the equipment states like starved, blocked, etc.
Equipment\Cycle Time\Overall		**Overall Cycle Time is the cycle including states like downtime, starved, blocked, etc.**
Average Overall Cycle Time	Float8	Average Overall cycle time in seconds for the time period selected
Max Overall Cycle Time	Float8	Max Overall cycle time in seconds for the time period selected
Min Overall Cycle Time	Float8	Min Overall cycle time in seconds for the time period selected
Overall Cycle Time	Float8	Overall Cycle Time in seconds is the cycle including states like downtime, starved, blocked, etc.
Equipment\Cycle Time\Precise		**Precise Cycle Time is the cycle time ignoring all the equipment states**
Average Precise Cycle Time	Float8	Average Precise cycle time in seconds for the time period selected
Max Precise Cycle Time	Float8	Max Precise cycle time in seconds for the time period selected
	Float8	

Data Point	Data Type	Description
Min Precise Cycle Time		Min Precise cycle time in seconds for the time period selected
Precise Cycle Time	Float8	Precise cycle time in seconds excluding states like planned downtime, unplanned downtime, starved and blocked.

Line Data Points

The Line Data points returns data for the line regardless of the Equipment the Live Analysis has been set up for.

Data Point	Data Type	Description
Line /Downtime		
Line Downtime Equipment Name	String	Name of the equipment that is responsible for causing line downtime
Line Downtime Equipment Path	String	Production model equipment path for equipment that is responsible for causing line downtime
Line Downtime Event Sequence	Int4	Every downtime event on the line is provided with an incrementing sequence number
Line Downtime Note	String	Note entered at the Line level

Data Point	Data Type	Description
Line Downtime Occurrence Count	Int4	Number of downtime events for the selected period.
Line Downtime Reason	String	The line or cell group (sub line) downtime reason. 1. When the line is down the Line Downtime Reason is the same as the Line State Name. 2. When the line is up the Line Downtime Reason is blank.
Line Downtime Reason Path	String	The full reason name for line or cell group (sub line) downtime reason. Line State name including State Class i.e. *Default/Cell Faulted*
Line Downtime Reason Split	Boolean	The line downtime reason split indicator. True is current downtime event has been split into multiple downtime events
Line Downtime State Time Stamp	DateTime	The time stamp for the equipment state change of the cell group (sub line) or cell that caused the line down time even.
Line /Meantime		
Line MTBF	Float8	The calculated Meantime (minutes) Between Failure for the selected period. Refer to Setting Up Equipment States - Meantime Metrics for more details.

Data Point	Data Type	Description
Line Meantime Metrics Enabled	Boolean	Returns if Meantime metrics have been enabled for this equipment.
Line /Schedule		
Line Schedule Available	Boolean	True if this operation was scheduled
Line Schedule Available Time	Float8	Time in minutes for available production time adjusted for line schedule availability and mode.
Line Standard Count	String	Amount of product that should have been produced based on the line schedule available time and line standard rate
Line Standard Count Variance	String	Variance between standard count and actual count
Line Target Count	String	Amount of product that should have been produced based on the line schedule available time and line schedule rate
Line Target Count Variance	String	Variance between line scheduled count and line OEE outfeed count.
Schedule Rate	Float8	See Schedule Rate for more details

Data Point	Data Type	Description
Line/State		
Line State Duration	Float8	The line or cell group (sub line) downtime event duration in minutes.
Line State Event Begin	DateTime	The line or cell group (sub line) downtime event begin date time.
Line State Event End	DateTime	The line or cell group (sub line) downtime event end date time.
Line State Event Sequence	Int4	The equipment state event sequence number.
Line State Name	String	The line or cell group (sub line) state. 1. When the line is down the Line Downtime Reason is the same as the state name. 2. When the line is up the Line Downtime Reason is blank.
Line State Override Scope	String	The state override scope for a line or cell group (sub line). See Setting Up Equipment - Override Scope for more details
Line State Override Type	String	The state override type for a line or cell group (sub line). See Setting Up Equipment - Override for more details
Line State Type	String	The line or cell group (sub line) state type. See Setting Up Equipment - State Type for more details

Data Point	Data Type	Description
Line State Value	Int4	The line or cell group (sub line) downtime state code. See Setting Up Equipment - State Code for more details

Equipment Mode & State Data Points

Data Point	Data Type	Description
Equipment /Mode		
Equipment Mode Name	String	Name of the current mode. See Setting Up Equipment Modes for more details
Equipment Mode Type	String	Name of the current mode type. See Setting Up Equipment Modes for more details
Equipment Mode Value	Int4	Name of the current mode. See Setting Up Equipment Modes for more details
Mode Begin Time	DateTime	Start time of the current mode
Mode Duration	Float8	Duration of the current mode in minutes
Mode End Time	DateTime	End time of the current mode
OEE Enabled	Boolean	See Setting Up Equipment Modes - OEE Enabled for more details
Production Counts Enabled	Boolean	See Setting Up Equipment Modes - OEE Enabled for more details

Data Point	Data Type	Description
Equipment /State		
Equipment Original State Value	Int4	The original value of equipment state tag collector before it is updated by using MES Value Editor component or scripting
Equipment State Name	String	Current state name
Equipment State Path	String	Production model equipment path for equipment that is responsible for causing line downtime
Equipment State Split	Boolean	True is current downtime event has been split into multiple downtime events
Equipment State Type	String	See Setting Up Equipment - State Type for more details
State Begin Time	DateTime	Start time of the current state
State Duration	Float8	Duration of current state in minutes
State End Time	DateTime	End time of the current state

Equipment Meantime Data Points

Data Point	Data Types	Description
Equipment /Meantime		

Data Point	Data Types	Description
Equipment MTBF	Float8	The Mean Time (minutes) Between Failure for the selected period
Equipment Meantime Metrics Enabled	Boolean	True if Equipment Meantime Metrics are enabled for the current equipment state

Equipment General Data Points

Data Point	Data Types	Description
Equipment /General		
Delta Time Stamp	Float8	Time gap (minutes) between the rows of data.
From Time Stamp	DateTime	Start time (minutes) of the current period
Shift	String	Name of the current shift as set by the Ignition Schedule Management component and defined for the current line or by the value passed in the equipment shift tag collector
Shift Day Text	String	Name of the current day
Shift Day of Month	Int4	Int value of the current month
Shift Day of Week	Int4	Int value of the current day of the week
Shift Day of Year	Int4	Int value of the current day of the year

Data Point	Data Types	Description
Shift ISO Week of Year	Int4	Int value of the ISO week of the year
Shift Month Text	String	Name of the current month
Shift Month of Year	Int4	Int value of the current month of the year
Shift Start Date	DateTime	Start time of the current shift
Shift Week of Month	Int4	Int value of the current week of the month
Shift Week of Year	Int4	Int value of the current week of the year
Shift Year	Int4	Int value of the current year
To Time Stamp	DateTime	Endtime (minutes) of the current period

Equipment OEE Data Points

Data Point	Data Type	Description
Equipment/OEE		
Elapsed Time	Float8	Elapsed Time of current operation
OEE	Float8	OEE value for selected period

Data Point	Data Type	Description
OEE General Count	Long	Any count value other than infeed, outfeed, reject and waste value for the selected time period
OEE Infeed Count	Long	Equipment infeed count value for the selected period
OEE Infeed Count Equipment Path	String	Infeed count tag collector path
OEE Outfeed Count	Long	Equipment outfeed count value for the selected period
OEE Outfeed Count Equipment Path	String	Outfeed count tag collector path
OEE Reject Count	Long	Equipment reject count value for the selected period
Planned Downtime	Float8	Planned Downtime duration (Double) for selected period
Runtime	Float8	Planned Downtime duration (Double) for selected period
Short Stop Time	Float8	Short stop duration (Double) for selected period
Standard Rate	Float8	See standard rate for more details
Target Changeover Time	Float8	Amount of time in minutes set for Target Changeover. See Changeover Duration for more details
Unplanned Downtime	Float8	Unplanned Downtime duration (Double) for selected period

Data Point	Data Type	Description
Equipment/OEE /Availability		
Is Short Stop	Boolean	True if current equipment state is consider a shortstop. See Short Stop Threshold for more details
OEE Availability	Float8	OEE Availability value for selected period
Equipment/OEE /Performance		
OEE Performance	Float8	OEE Performance value for selected period
Infeed Standard Count	Float8	Calculated expected infeed based on standard rate
Equipment/OEE /Quality		
OEE Quality	Float8	OEE Quality value for selected period

Setting Values

The analysis results that are returned can be modified through the use of settings. Setting values provide a number of keywords as listed below.

Format for entering the keywords is k*eyword1=True, keyword2=100.0, keyword3=10.*

> ⚠ Settings like Enable Totalized Mode, Include Future, Last Values and Rollup Time span is meant for analysis selector and not for live analysis

Setting	Description	Use	Example
Date Format		All	

Setting	Description	Use	Example
	Date format fields can be customized with this setting e.g. 'YYYY/MM/dd hh:mm:ss a'		Date Format = `2017/04/12 19:45:30`
Enable Totalized Mode	This setting accumulates the count. Useful for charts where you wish to display the accumulated production count over time	Not valid for Live Analysis	Enable Totalized Mode = True
Include Future	Allows for count values to be calculated in the future. Useful for charts where you want to display target counts for future runs	Not valid for Live Analysis	Include Future = True
Last Values	Only the latest values are shown.	Not valid for Live Analysis	Last Values = True
OEE Availability Cap	The maximum value calculated can be capped with this setting	All	OEE Availability Cap = 100.0
OEE Performance Cap	The maximum value calculated can be capped with this setting	All	OEE Performance Cap = 100.0
OEE Quality Cap	The maximum value calculated can be capped with this setting	All	OEE Quality Cap = 100.0
Rollup Time Span	If the time (seconds) between downtime events is less than the rollup time and it is the same equipment and reason,	Not valid for Live Analysis	Rollup Time Span = 30

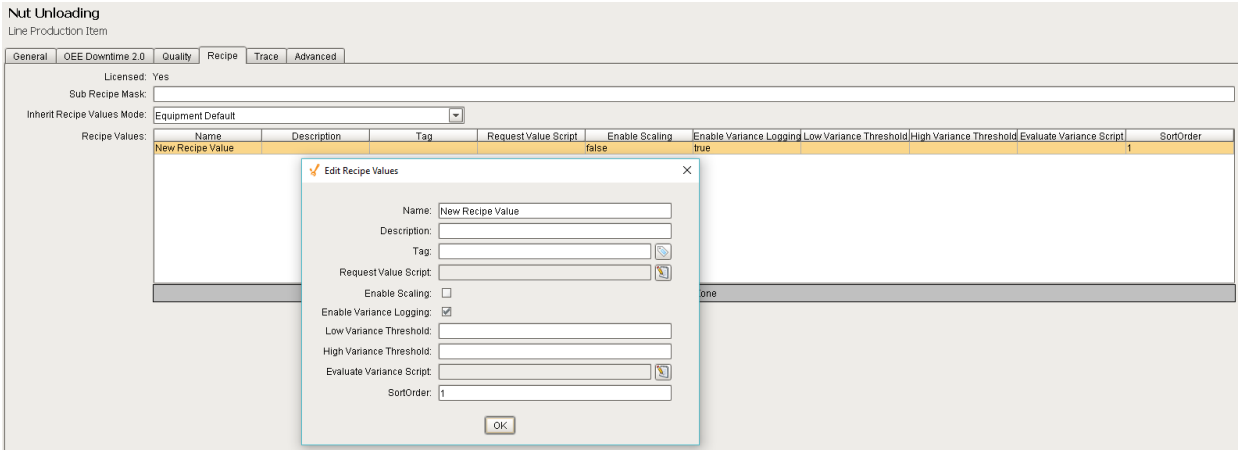
Setting	Description	Use	Example
	then it will rollup the event into one row in the results and will increase the occurrence count.		
Row Limit	The analysis can be limited to a certain number of rows.	All	Row Limit = 10

7.8.4 Recipe Tab

Recipe Values names can be inherited from the Enterprise, Site or Area level settings, or defined at the Line level.

At this level, it is possible to add configuration information regarding the tag, variance thresholds and the scripts used to evaluate them.

For more information please refer to the Recipe Production Model Configuration section.

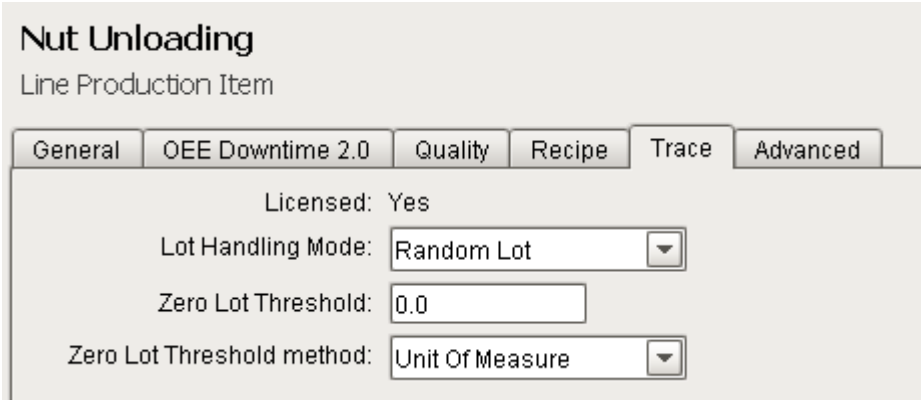


Recipe Tab

7.8.5 Trace Tab

The Trace tab allows you to define the **Lot Handling Mode** for the selected line.

Refer to the Lot Handling Mode section in the Track & Trace Module help.

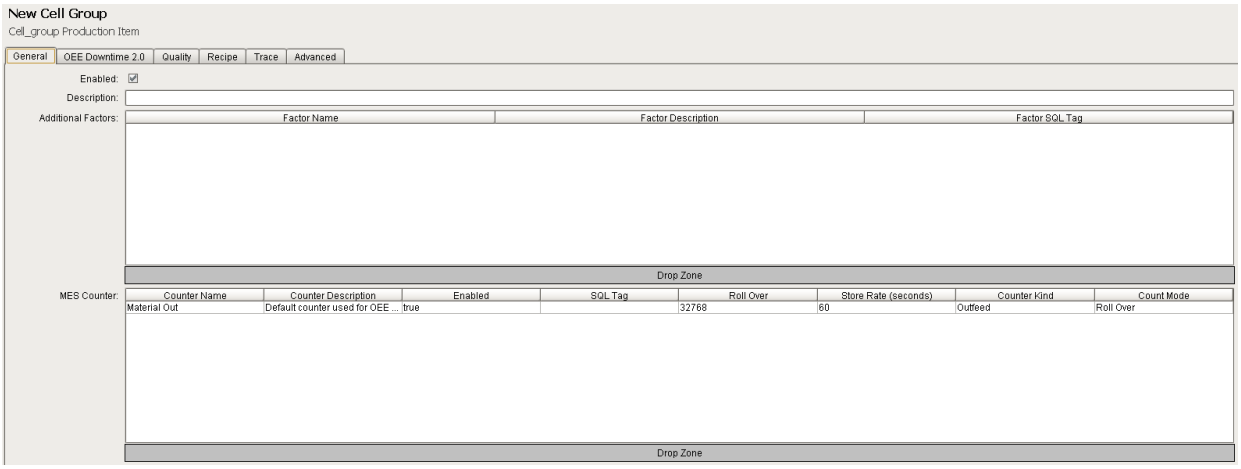


Trace Tab

7.9 Cell Group Settings

The Cell Group Production Item provides a configuration page for setting up a Cell Group for all the MES modules. The General and Advanced Tab are always present whereas each module when installed provides an additional tab.

Cell groups provide a mechanism for creating sections of a production line that may contain parallel process cells or to model sub-lines that feed into a main production line.



7.9.1 General Tab - Additional Factors

The OEE Module collects and logs a number of downtime and production data values. However, what if other values outside of downtime and production values are of interest? Additional factors are the solution. Additional Factors are user defined data points that are logged along with the production and downtime information. Once they are logged, they can be shown in charts, tables and reports. Additionally, other analysis can be done by filtering and/or setting up comparisons by their values.

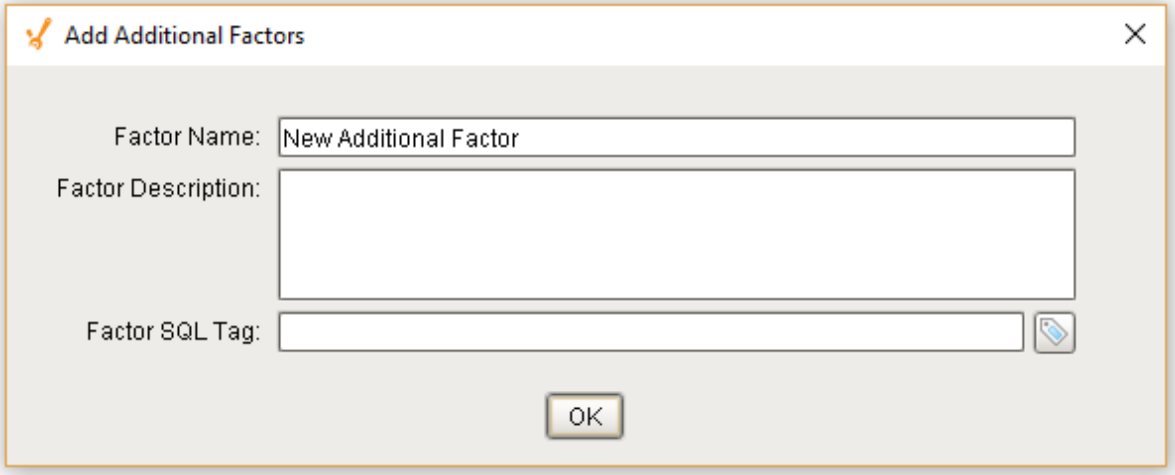
Additional Factors can added to the Line, Cell Group and Cell Production Items in the Production Model Designer.

Any value that can be read from an Ignition SQLTag can be added as an additional factor. This includes values derived from scripts, or from barcode readers, databases, calculations, PLCs, etc.

Any tag can be added as an additional factor. To configure, select a Line in the production model and select the **General** tab on the right. Right click on the **Additional Factors** table and select **New**. An additional factor is simply just a name and a tag.

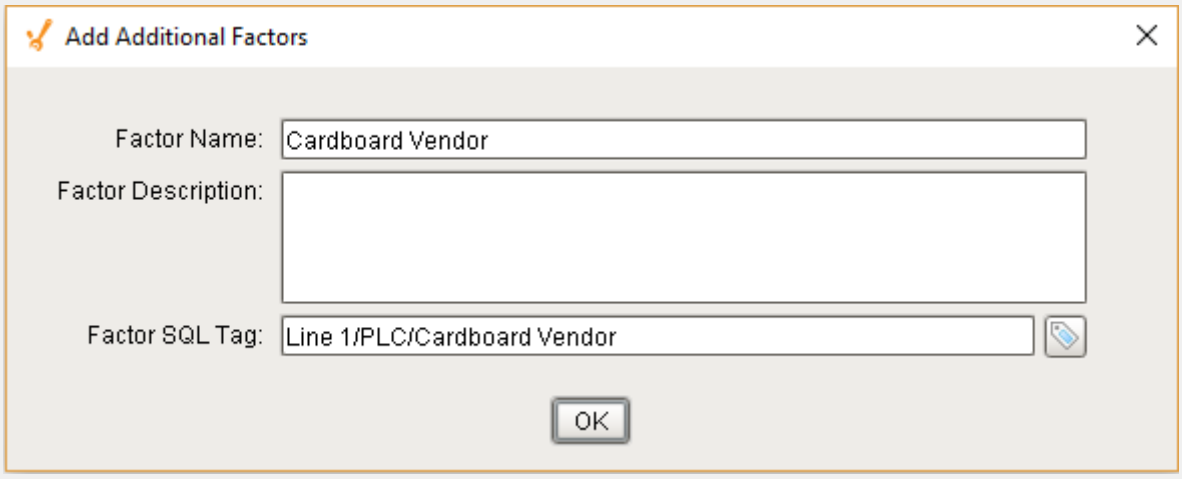
Properties

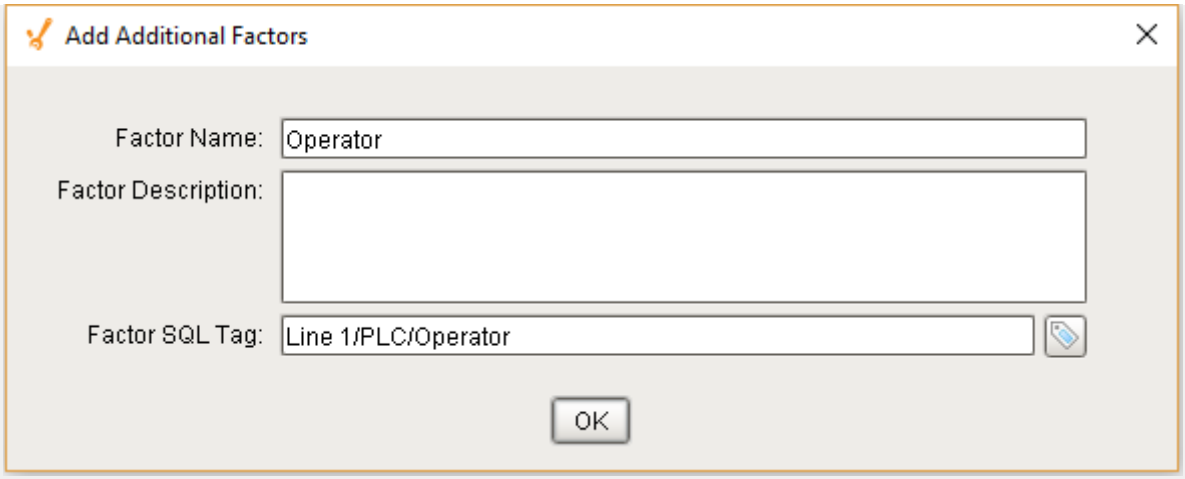
Factor Description	Optionally, this property can be set to a description for the additional factor. It is not used by the OEE Downtime and Scheduling Module other than for reference.	String
Factor Name	This reflects the name of additional factor that is configured in the designer.	String Read Only
Factor SQLTag	This reflects the Factor SQLTag setting that additional factor is configured for in the designer. It is the name of SQLTag to read the factor value from.	String Read Only



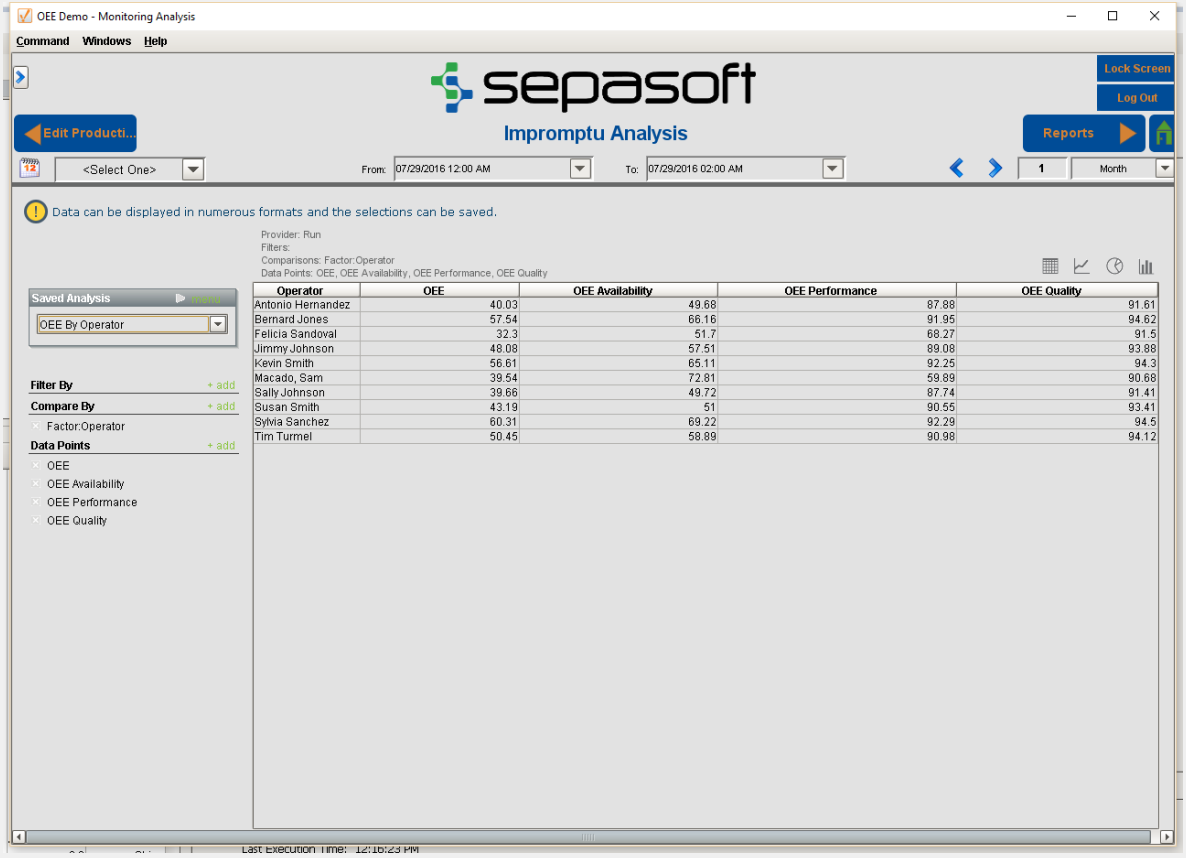
Example

In the example, we have two factors, **Cardboard Vendor** and **Operator**. The operator can select the vendor that provided the cardboard or it can be obtained from some other source. Now, OEE and downtime results can be shown for each cardboard manufacturer. This can identify quality problems with raw material that directly affect production efficiency. With the operator setup as an additional factor, the operator's name will be logged along with the production and downtime data. By doing so, OEE and downtime information can be filtered and grouped by the operator name. But this could just as well be the production crew, supervisor, maintenance crew or any other user defined value that can be monitored or entered into the system.





Adding these factors to a production line will allow us to capture the value of these tags whenever they change. In the impromptu analysis, we can then compare OEE values by our additional factor: Operator



7.9.2 General Tab - MES Counters

The MES Counters are used to associate Process Segments (Operations) with production counts. MES Counters record production counts 7/24, independent of scheduled production runs.

Counter names and the associated tag are defined in the Production Model. In the MES Management screen, the Quantity Source of Infeed and Material Process Segments can be set to use these MES counters.

MES counters are available for the Line, Cell Group, Cell or a Storage Unit production items in the Production Model Designer.

It is recommended to set up MES Counters at the cell level in order to accurately capture counts while indexing product on the line. Addition configuration must be accomplished with the cell settings in the production model.

The quantity from the MES counters can be obtained through scripting, see system.mes.getCountValue.

Adding MES Counter

To configure an MES Counter, select a Line, Cell Group, Cell or a Storage Unit in the production model and select the **General** tab on the right. Right click on the **MES Counter** table and select **New**. Properties can be set through the **Add MES Counter** Window.

Counter Name

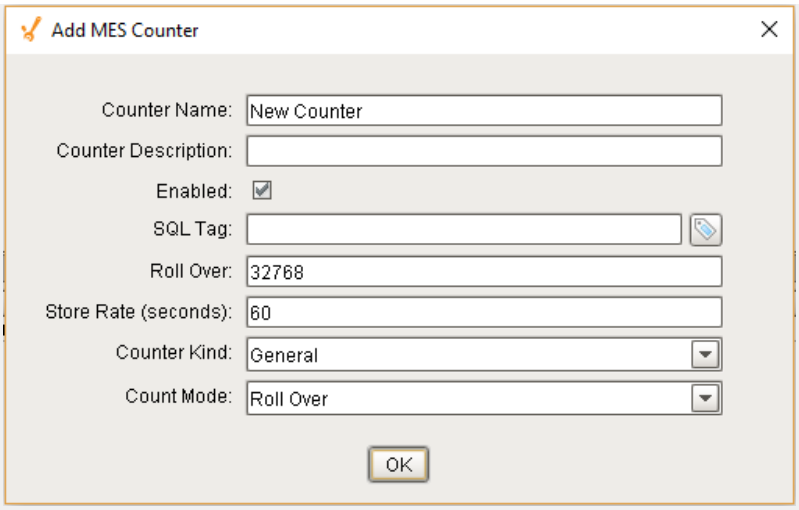
Name of the counter.

Counter Description

The description for the MES counter. This setting is not mandatory.

Enabled

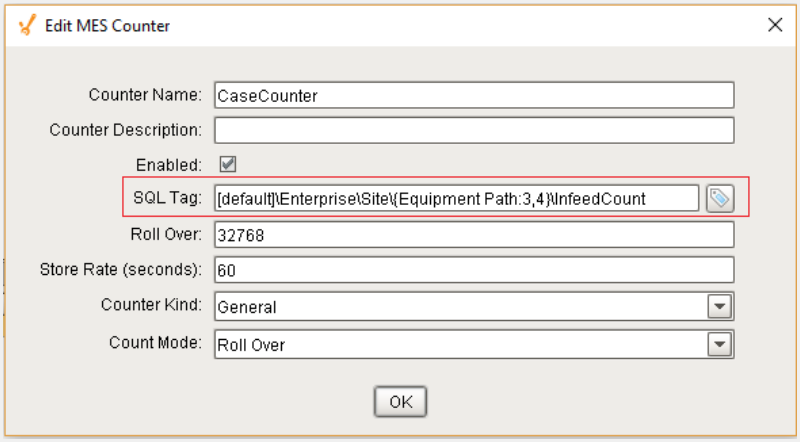
The counter can be enabled or disabled here.



SQL Tag

The path to the Tag Provider and ignition tag where the count value will come from is assigned to the MES counter here.

Parameterized Tag Paths can be used here which allows for indirection and exported MES Counters to be easily deployed to other equipment.



Roll Over

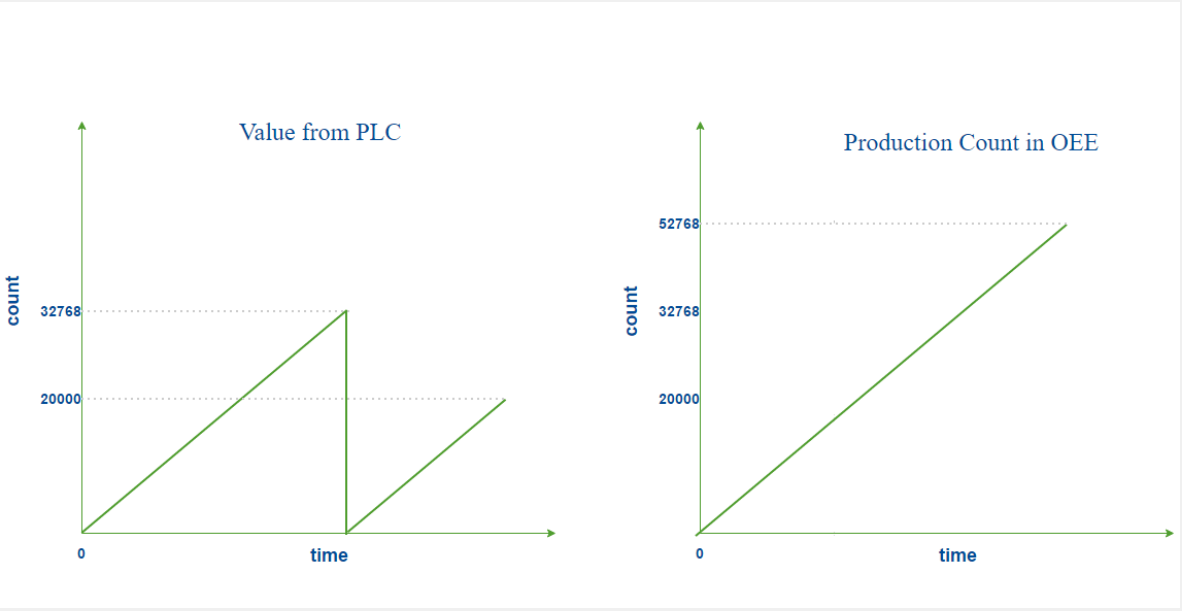
For PLC count tags that that do not get reset, they will eventually reach a finite maximum value at which point, the value will 'rollover' back to zero. The Roll Over setting allows you to define the value that should be added to the count tag whenever a roll over occurs. By default it is 32768 which equates to a 16 bit signed data value. Your setting will be dependent upon the datatype of your plc count tag.

During a production run, the incoming count value is added to the Roll Over setting multiplied by the number of times a rollover has occurred.

Example: production count value = incoming count value + 32768 * 3

Production Count	PLC Count tag	Rollover	Calculation
32700	32700	0	32700 + 32768 * 0
32750	32750	0	32700 + 32768 * 0
32918	150	1	150 + 32768 * 1
33068	300	1	300 + 32768 * 1

⚠ This value is only used when the **Count Mode** is set to **Rollover**.



Roll Over Count Mode

Store Rate

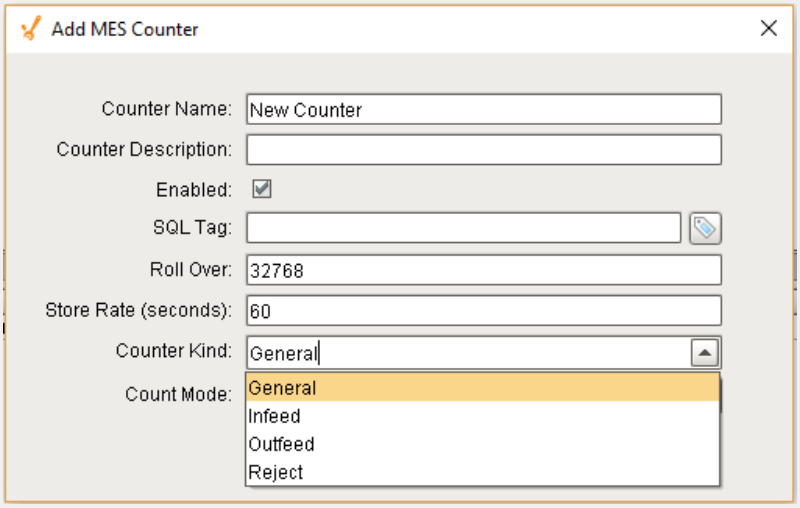
The MES counter will be captured and stored in the database after this specific interval in seconds if the value has changed If **Store Rate** is set to zero, every value change will be recorded.

Counter Kind

MES Counters can be set to four different kinds:

- Infeed
- Outfeed
- Reject
- General

Infeed, **Outfeed** and **Reject** kinds are used solely by the OEE module to determine which MES counter to use for OEE Performance, OEE Quality and production count information. The **General** kind can be used for any other count value.



Count Mode

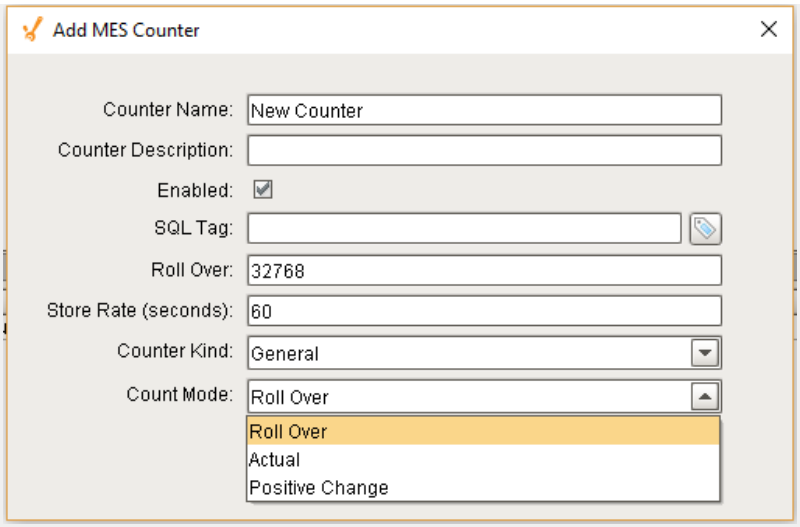
The Count Mode can be set to **Roll Over**, **Actual** and **Positive Change**.

Roll Over

See Rollover section for this count mode.

Actual

The Actual count mode simply uses whatever value is passed through the sql tag to represent the actual production counts. Production counts can go down as well as up.

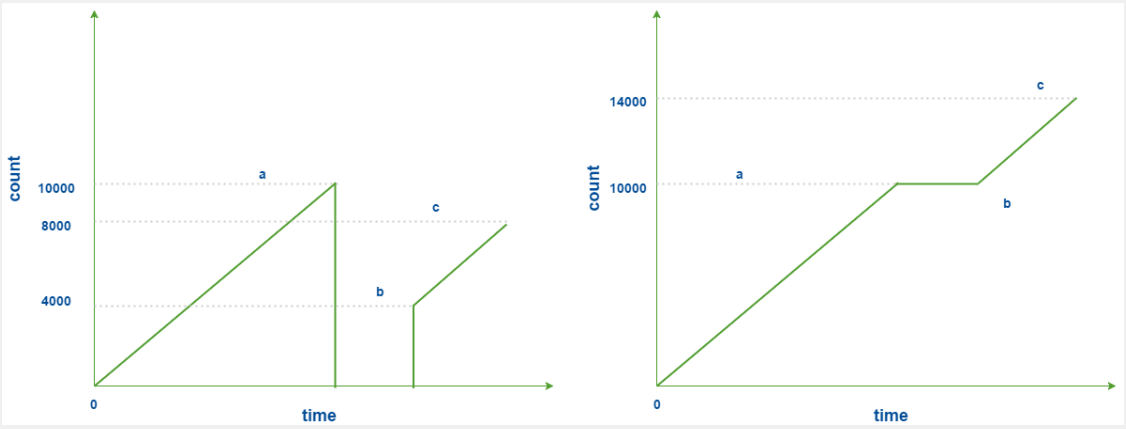


Positive Change

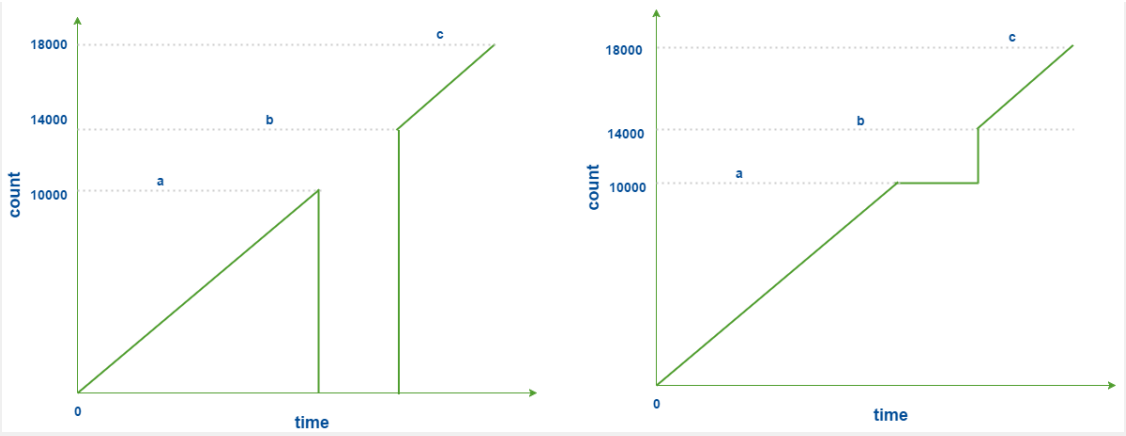
The Positive Change mode ignores any sql tag count values that are zero and will accumulate the counts. Three different cases are illustrated using the graphs shown.

PLC Count vs. OEE Count

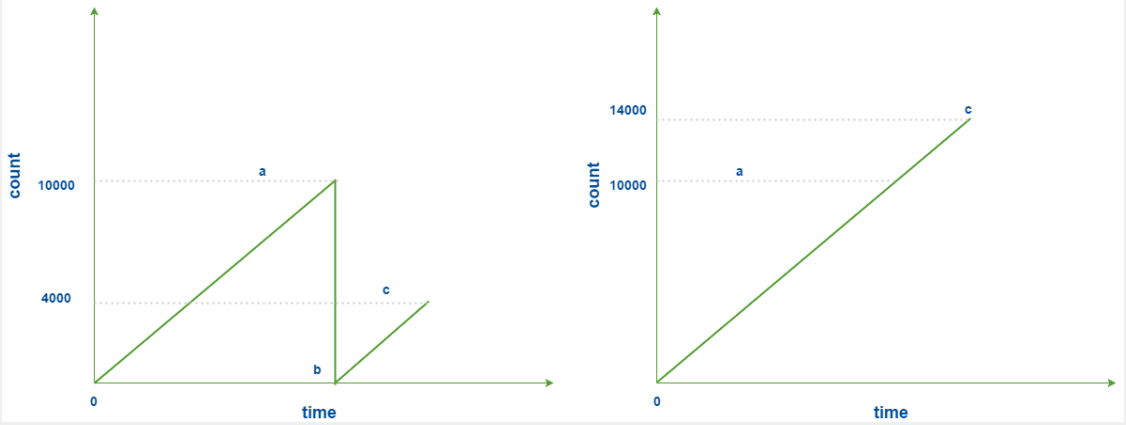
Case 1



Case 2



Case 3



Counter Rapid Development Features

Not only do counters allow for parameterization (as shown above), they also support copy, paste, import, and export features for rapid development. Configure the infeed counter for one Production Model Node (i.e. a Cell or Line) with parameterization, then copy and paste it to the other cells. Alternatively, copy and paste the Node itself and rename it for a similar effect.

7.9.3 OEE 2.0 Downtime Tab

The OEE Downtime 2.0 tab is specific to the OEE module and is available for the Line, Cell Group and Cell Production Items. A number of configuration settings are provided that can be used to obtain equipment mode, state and count values from ignition tags (whether plc tags, memory or expression tags).

Downtime Detection Mode

How line downtime is determined can be changed based on the selected Downtime Detection Mode. Valid options for the downtime detection mode are...

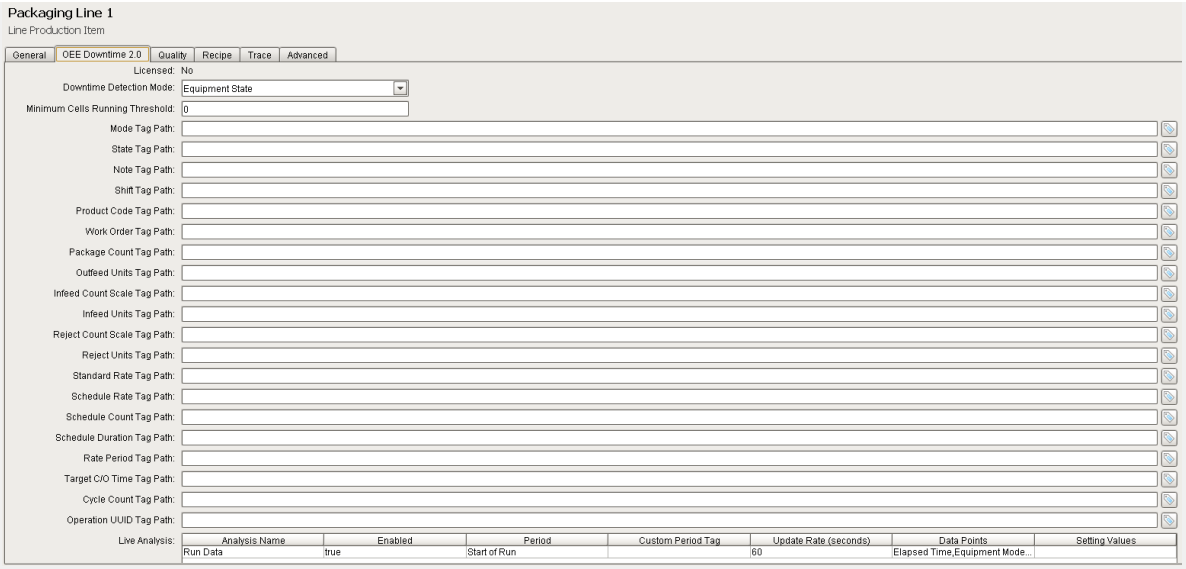
- Equipment State
- Key Reason (Cell Priority)
- Key reason (Neighbor Priority)
- Initial Cell
- Parallel Cells

Refer to Downtime Detection Mode for more information on the various Downtime Detection Methods.

> ⚠ Downtime Detection Mode is only available for the Line and Cell Group Production Item.

Minimum Cells Running Threshold

Minimum Cells Running Threshold determines how many cells in the Line (or Cell Group) must be running in order for the Line (or Cell Group) to be considered as Running.



Tag Collector Paths

Tag Collectors are provided to allow any of the parameters needed to drive OEE Metrics to be provided externally to the OEE module. Virtually all the Tag Collectors can be left blank in which case the OEE engine will determine the value from product code configuration information as defined in the OEE Material Manager or from internal calculations. Exceptions to this would be the equipment state and counts where needed.

> When adding a tag to a Tag Collector, you must use memory tags. The Production Model will write values to any tags defined here as well as read the value whenever it changes from an external source.

Tag Collector Path	Tag Type	Data Type	Description
Mode	Memory	Integer	The Mode Tag, if provided, will be written to by the Production Model whenever the equipment mode changes based on Material Production Settings. The value of the mode tag can also be written to whenever the Mode value changes either indirectly from a plc tag (via tag change event) or from the HMI. The value of the Mode tag will be recorded for the current mode.
State	Memory	Integer	

Tag Collector Path	Tag Type	Data Type	Description
			The State Tag path will generally come from a plc as the source of the current equipment state. Exceptions to this are at the Line level when using a downtime detection method other than Equipment State.
Downtime Note	Memory	String	Apart from scripting and Downtime table component, the downtime notes can be added by using tags.
Shift	Memory	String	When left blank, shifts defined in the Ignition Schedule Management component and defined in the Equipment Manager for a line will be used to determine the current shift. If a tag is provided here, whatever value is in the tag e.g. 'Shift A' will be recorded for the current shift.
Product Code	Memory	String	When left blank, the Product Code currently running on the line will be determined from the scheduled run as selected by the Scheduler or Run Director component. When a tag path is provided, the product code for the line or equipment (cell) will be determined from the value of the tag.
Work Order	Memory	String	When left blank, the Work Order currently running on the line will be determined from the scheduled run as selected by the Scheduler or Run Director component. When a tag path is provided, the product code for the line or equipment (cell) will be determined from the value of the tag.
Package Count	Memory	Float	When left blank, the Package Count will be determined from the Package count setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Package Count for the line or equipment (cell) will be determined from the value of the tag. For more information on the Package Count, refer to the Material Production Settings section.

Tag Collector Path	Tag Type	Data Type	Description
Line Outfeed Units	Memory	String	When left blank, the Line Outfeed Units will be determined from the Line Outfeed Units setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Line Outfeed Units for the line or equipment (cell) will be determined from the value of the tag. For more information on the Line Outfeed Units, refer to the Material Production Settings section.
Infeed Count Scale	Memory	Float	When left blank, the Infeed Count Scale will be determined from the Infeed Count Scale setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Infeed Count Scale for the line or equipment (cell) will be determined from the value of the tag. For more information on the Infeed Count Scale, refer to the Material Production Settings section.
Line Infeed Units	Memory	String	When left blank, the Line Infeed Count Scale will be determined from the Infeed Count Scale setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Infeed Count Scale for the line or equipment (cell) will be determined from the value of the tag. For more information on the Line Infeed Units, refer to the Material Production Settings section.
Reject Count Scale	Memory	Float	When left blank, the Reject Count Scale will be determined from the Reject Count Scale setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Reject Count Scale for the line or equipment (cell) will be determined from the value of the tag. For more information on the Reject Count Scale, refer to the Material Production Settings section.

Tag Collector Path	Tag Type	Data Type	Description
Line Reject Units	Memory	String	When left blank, the Line Reject Units will be determined from the Line Reject Units setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Line Reject Units for the line or equipment (cell) will be determined from the value of the tag. For more information on the Line Reject Units, refer to the Material Production Settings section.
Standard Rate	Memory	Float	When left blank, the Standard Rate will be determined from the Standard Rate setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Standard Rate for the line or equipment (cell) will be determined from the value of the tag. For more information on the Standard Rate, refer to the Material Production Settings section.
Schedule Rate	Memory	Float	When left blank, the Schedule Rate will be determined from the Schedule Rate setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Schedule Rate for the line will be determined from the value of the tag. For more information on the Schedule Rate, refer to the Material Production Settings section. ⚠ Schedule Rate Tag Path is only available for the Line Production Item.
Schedule Count	Memory	Integer	When left blank, the Schedule Count will be determined from the scheduled run as selected by the Scheduler or Run Director component. When a tag path is provided,

Tag Collector Path	Tag Type	Data Type	Description
			the Schedule Count for the line or equipment (cell) will be determined from the value of the tag. The Schedule Count provides the number of units scheduled to be produced. ⚠ Schedule Count Tag Path is only available for the Line Production Item.
Schedule Duration	Memory	Integer	When left blank, the Schedule Duration will be determined from the scheduled run as selected by the Scheduler or Run Director component. When a tag path is provided, the Schedule Duration for the line or equipment (cell) will be determined from the value of the tag. The Schedule Duration provides the expected runtime required for the number of units scheduled to be produced and is calculated by the Schedule Rate.
Rate Period	Memory	String	When left blank, the Rate Period will be determined from the Rate Period setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Rate Period for the line or equipment (cell) will be determined from the value of the tag. For more information on the Rate Period, refer to the Material Production Settings section.
Target C /O Time	Memory	Integer	When left blank, the Target C/O (Changeover) Time will be determined from the Changeover settings for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Target C/O (Changeover) Time for the line or equipment (cell) will be determined from the value of the tag. For more information on the Target C/O (Changeover) Time, refer to the Material Production Settings section.

Tag Collector Path	Tag Type	Data Type	Description
			Target C/O Time Tag Path is only available for the Line Production Item.
Cycle Count	Memory	Float	When left blank, the Cycle Count will be determined by the OEE Module. When a tag path is provided, the Cycle Count for the line or equipment (cell) will be determined from the value of the tag.
Operation UUID	Memory	String	When left blank, the Operation UUID will be determined from the currently running Operation on the Line or equipment (cell). When a tag path is provided, the Operation UUID for the line or equipment (cell) can be determined from the value of the tag. The purpose for this tag is to be able to provide OEE analysis data when production runs are not scheduled or started using the Run Director or Schedule Selector components, or scripting functions. In this case a tag can be used to provide a Run Identifier value i.e. Run_4253_XX. The Analysis Selector provides the ability to pull the Operation UUID as part of analysis, whether it is an internally generated Operation UUID or a passed Run Identifier.

Tag Collector Paths can be parameterized with {Equipment Path} to utilize indirection and more rapidly implement the production model. See Parameterized Tag Paths for more details.

Live Analysis

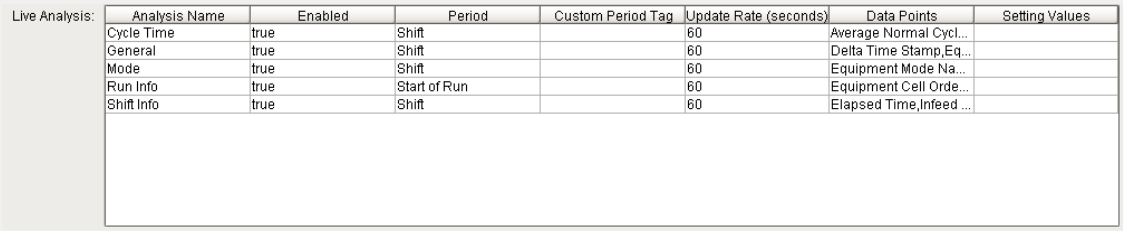
Live Analysis provides a flexible way of customizing your application to provide a set of real-time tag values that can be accessed from the Ignition designer and used in your application to provide real-time production monitoring. Live Analysis is configured

in the OEE 2.0 Downtime tab of the Production Model Designer for the Line, Cell Group and Cell production items. When a Live Analysis is created, a corresponding set of tags is created in the MES Tag Provider that provide the real-time status of those datapoints based upon the Period defined for the Live Analysis. You can create multiple Live Analysis and use those tags to drive HMI displays.

To create a new Live Analysis:

- Right click on the Live Analysis panel on the OEE 2.0 Downtime Tab in the Production Model Designer.
- Provide a Name
- Select the Period that the Live Analysis datapoints will return a value for. Valid options are Shift, Day (Midnight), Day (Production), Start of Run, Top of Hour, Custom Period Tag
- Select the frequency for how often the tag values will be updated. Default value is 60 seconds. Minimum value is 10 seconds
- Select the desired Data Points
- Add any further Settings Values required

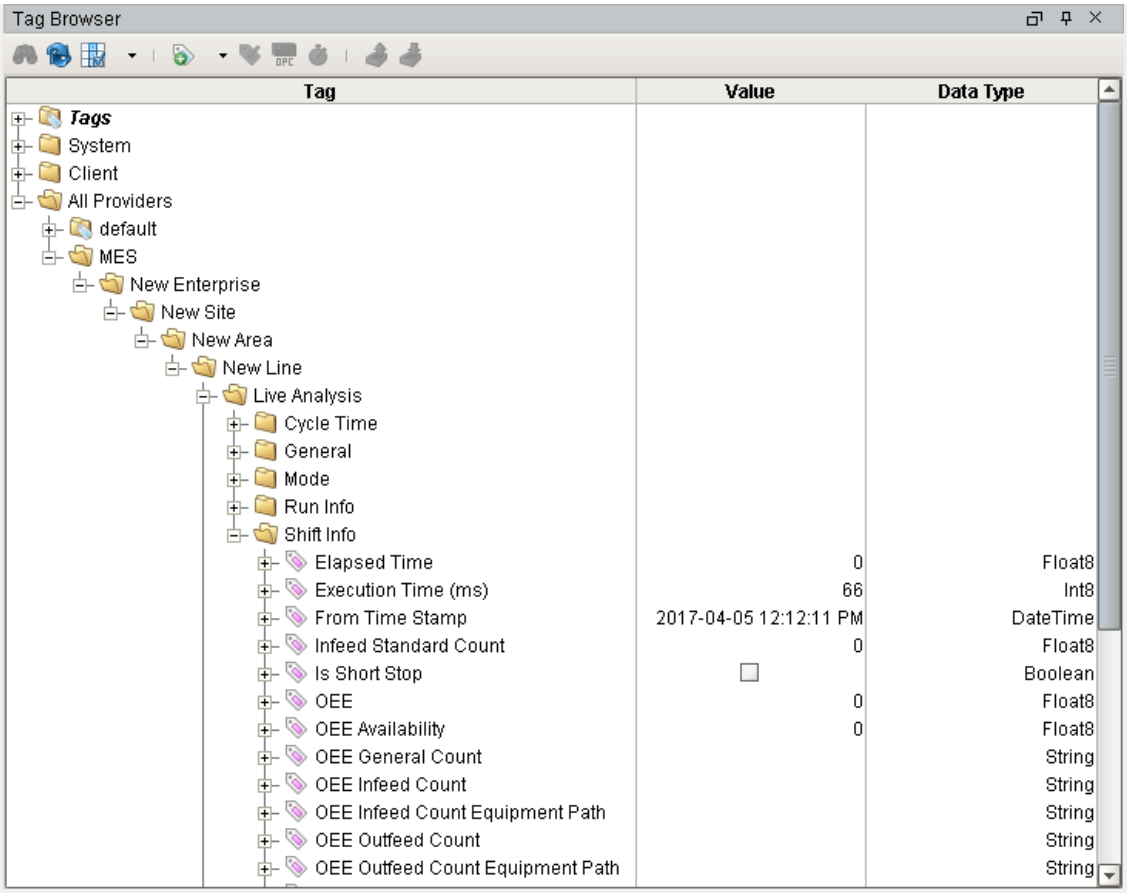
You cannot select all Data Points in one Live Analysis. The maximum length string for Data Points is 1024 characters



Live Analysis:

Analysis Name	Enabled	Period	Custom Period Tag	Update Rate (seconds)	Data Points	Setting Values
Cycle Time	true	Shift		60	Average Normal Cycl...	
General	true	Shift		60	Delta Time Stamp,Eq...	
Mode	true	Shift		60	Equipment Mode Na...	
Run Info	true	Start of Run		60	Equipment Cell Orde...	
Shift Info	true	Shift		60	Elapsed Time,Infeed ...	

Live Analysis Settings Panel in the OEE 2.0 Downtime Tab



MES Tag Provider Live Analysis Tags

Live Analysis Settings

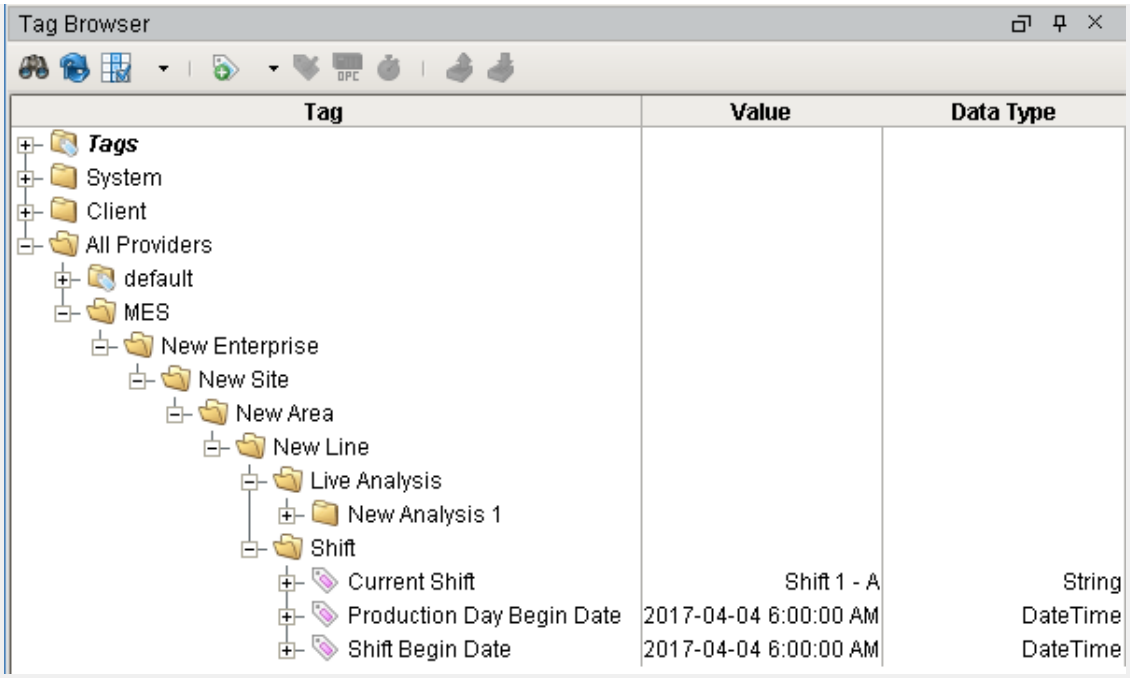
Setting	Description
Analysis Name	The name for the live analysis.
Enabled	The live analysis can be enabled or disabled with this setting.
Period	The duration of analysis can be set by **Shift**, **Day (midnight)**, **Day (production)**, **Start of Run**, **Top of Hour** or **Custom Period Tag**.
Custom Period Tag	A tag can be assigned to define the start datetime for a custom period. The end time will be the current time. It takes value in the date time data type. Example for a valid value for the custom period tag is: **`2017/04/04 14:00:00`**

Setting	Description
Update Rate	The rate in seconds by which the live analysis is updated. The minimum update rate is 60 seconds.
Data Points	Data points allows you to pick and choose the values you wish to access through tags. See the table below for the listing of available data points.

Shift Data Points

When creating a Live Analysis, the following shift data points will be automatically created.

Data Point	Data Type	Description
Current Shift	String	The currently running shift as defined in the Ignition Schedule Management component or passed from the Shift Tag Collector path
Production Day Begin Date	DateTime	Production start time
Shift Begin Date	DateTime	Start time of the current shift



Analysis Data Points and Settings are used by Live Analysis, the MES Analysis Selector and MES Analysis Controller components, the MES Analysis Data Source for reporting and the MES Analysis Settings object.

Equipment Data Points

Data Point	Data Type	Description
Equipment		
Equipment Cell Order	Int4	Integer value that determines the cell order of the equipment within the line. Is set to *null* for the line. Is set to 0 for first cell within each cell group
Equipment Name	String	Name of the equipment as defined in the production model
Equipment Note	String	Any note that has been recorded for this piece of equipment through the Note tag collector path in the Production model will be exposed here.
	DateTime	

Data Point	Data Type	Description
Equipment Operation Begin		Start Date time of the currently running operation on this equipment
Equipment Operation Sequence	Int4	The ordinal number (integer) of operation
Equipment Path	String	Production model path for this equipment
Equipment Type	String	Can be Line, Cell Group or Cell
Execution Time (ms)	Int8	Time taken to execute and update the Live Analysis. Used mainly for performance debugging
From Time Stamp	DateTime	Start Date Time of current data point results
Infeed Units	String	See Infeed Units for more details
Is Key Cell	Boolean	See Key Reason for more details
Operation UUID	String	Unique Identifier for currently running operation
Outfeed Units	String	See Outfeed Units for more details
Product Code	String	Product code currently being processed on this equipment
Rate Period	String	See Rate Period for more details

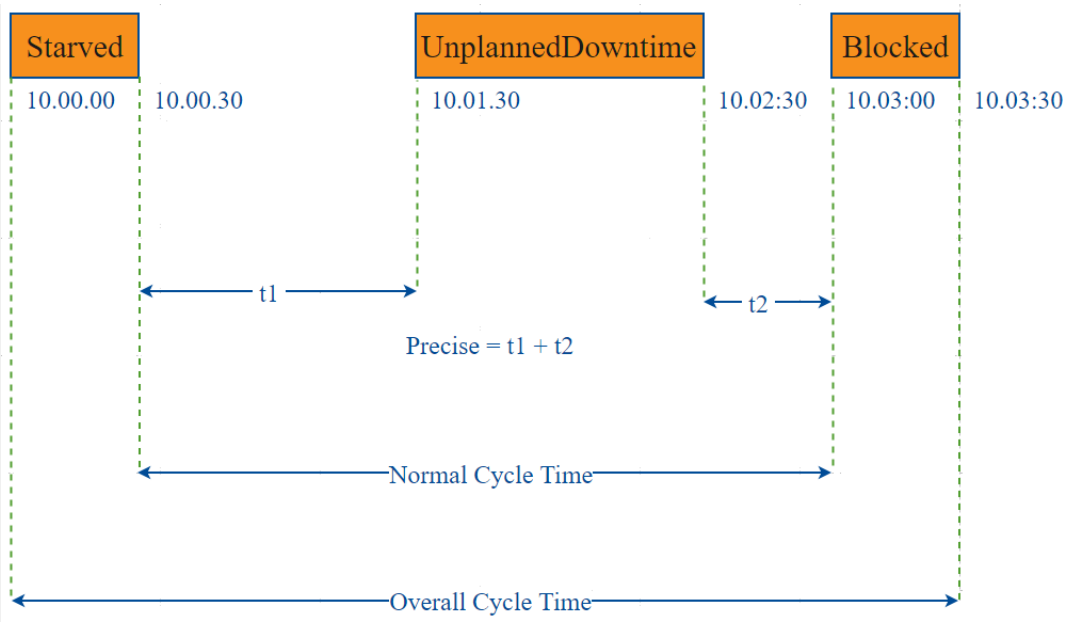
Data Point	Data Type	Description
Reject Units	String	See Reject Units for more details
To Time Stamp	DateTime	End Date Time of current data point results
Work Order	String	Work order currently being processed on this equipment

Equipment Count Data Points

Data Point	Data Type	Description
Equipment\Count		*Any defined counters for the production item will also appear in this folder
Equipment Infeed Scale	Float8	See Infeed Count Scale for more details
Equipment Package Count	Float8	See Package Count for more details
Equipment Reject Scale	Float8	See Reject Count Scale for more details
Outfeed-Material Out	String	Value of the default MES Counter used for OEE outfeed count

Equipment Cycle Time Data Points

The Cycle Time data points provides a number of metrics that can be used to measure the amount of time required to produce one piece. It is often used to gain an understanding of variations in production. Live Analysis provides Target, Normal, Overall and Precise Cycle Time metrics.



Data Point	Data Type	Description
Equipment\Cycle Time		
Relative Cycle Count	String	Relative Cycle Count is how many occurred for the compare by.
Target Cycle Time	Float8	Also known as Takt time, it is how often a piece must be produced to meet customer demand. It is often used to pace a production line, and it is a calculated number in seconds.
Total Cycle Count	String	Total Cycle Count is accumulative, it is sum total of all the cycle count.
Equipment\Cycle Time\Normal		**Normal Cycle Time is the actual cycle ignoring the equipment states like starved, blocked, etc.**
Average Normal Cycle Time	Float8	Average Normal cycle time in seconds for the time period selected
	Float8	

Data Point	Data Type	Description
Max Normal Cycle Time		Max Normal cycle time in seconds for the time period selected
Min Normal Cycle Time	Float8	Min Normal cycle time in seconds for the time period selected
Normal Cycle Time	Float8	Normal Cycle Time in seconds is the actual cycle ignoring the equipment states like starved, blocked, etc.
Equipment\Cycle Time\Overall		**Overall Cycle Time is the cycle including states like downtime, starved, blocked, etc.**
Average Overall Cycle Time	Float8	Average Overall cycle time in seconds for the time period selected
Max Overall Cycle Time	Float8	Max Overall cycle time in seconds for the time period selected
Min Overall Cycle Time	Float8	Min Overall cycle time in seconds for the time period selected
Overall Cycle Time	Float8	Overall Cycle Time in seconds is the cycle including states like downtime, starved, blocked, etc.
Equipment\Cycle Time\Precise		**Precise Cycle Time is the cycle time ignoring all the equipment states**
Average Precise Cycle Time	Float8	Average Precise cycle time in seconds for the time period selected
Max Precise Cycle Time	Float8	Max Precise cycle time in seconds for the time period selected
	Float8	

Data Point	Data Type	Description
Min Precise Cycle Time		Min Precise cycle time in seconds for the time period selected
Precise Cycle Time	Float8	Precise cycle time in seconds excluding states like planned downtime, unplanned downtime, starved and blocked.

Line Data Points

The Line Data points returns data for the line regardless of the Equipment the Live Analysis has been set up for.

Data Point	Data Type	Description
Line /Downtime		
Line Downtime Equipment Name	String	Name of the equipment that is responsible for causing line downtime
Line Downtime Equipment Path	String	Production model equipment path for equipment that is responsible for causing line downtime
Line Downtime Event Sequence	Int4	Every downtime event on the line is provided with an incrementing sequence number
Line Downtime Note	String	Note entered at the Line level

Data Point	Data Type	Description
Line Downtime Occurrence Count	Int4	Number of downtime events for the selected period.
Line Downtime Reason	String	The line or cell group (sub line) downtime reason. 1. When the line is down the Line Downtime Reason is the same as the Line State Name. 2. When the line is up the Line Downtime Reason is blank.
Line Downtime Reason Path	String	The full reason name for line or cell group (sub line) downtime reason. Line State name including State Class i.e. *Default/Cell Faulted*
Line Downtime Reason Split	Boolean	The line downtime reason split indicator. True is current downtime event has been split into multiple downtime events
Line Downtime State Time Stamp	DateTime	The time stamp for the equipment state change of the cell group (sub line) or cell that caused the line down time even.
Line /Meantime		
Line MTBF	Float8	The calculated Meantime (minutes) Between Failure for the selected period. Refer to Setting Up Equipment States - Meantime Metrics for more details.

Data Point	Data Type	Description
Line Meantime Metrics Enabled	Boolean	Returns if Meantime metrics have been enabled for this equipment.
Line /Schedule		
Line Schedule Available	Boolean	True if this operation was scheduled
Line Schedule Available Time	Float8	Time in minutes for available production time adjusted for line schedule availability and mode.
Line Standard Count	String	Amount of product that should have been produced based on the line schedule available time and line standard rate
Line Standard Count Variance	String	Variance between standard count and actual count
Line Target Count	String	Amount of product that should have been produced based on the line schedule available time and line schedule rate
Line Target Count Variance	String	Variance between line scheduled count and line OEE outfeed count.
Schedule Rate	Float8	See Schedule Rate for more details

Data Point	Data Type	Description
Line/State		
Line State Duration	Float8	The line or cell group (sub line) downtime event duration in minutes.
Line State Event Begin	DateTime	The line or cell group (sub line) downtime event begin date time.
Line State Event End	DateTime	The line or cell group (sub line) downtime event end date time.
Line State Event Sequence	Int4	The equipment state event sequence number.
Line State Name	String	The line or cell group (sub line) state. 1. When the line is down the Line Downtime Reason is the same as the state name. 2. When the line is up the Line Downtime Reason is blank.
Line State Override Scope	String	The state override scope for a line or cell group (sub line). See Setting Up Equipment - Override Scope for more details
Line State Override Type	String	The state override type for a line or cell group (sub line). See Setting Up Equipment - Override for more details
Line State Type	String	The line or cell group (sub line) state type. See Setting Up Equipment - State Type for more details

Data Point	Data Type	Description
Line State Value	Int4	The line or cell group (sub line) downtime state code. See Setting Up Equipment - State Code for more details

Equipment Mode & State Data Points

Data Point	Data Type	Description
Equipment /Mode		
Equipment Mode Name	String	Name of the current mode. See Setting Up Equipment Modes for more details
Equipment Mode Type	String	Name of the current mode type. See Setting Up Equipment Modes for more details
Equipment Mode Value	Int4	Name of the current mode. See Setting Up Equipment Modes for more details
Mode Begin Time	DateTime	Start time of the current mode
Mode Duration	Float8	Duration of the current mode in minutes
Mode End Time	DateTime	End time of the current mode
OEE Enabled	Boolean	See Setting Up Equipment Modes - OEE Enabled for more details
Production Counts Enabled	Boolean	See Setting Up Equipment Modes - OEE Enabled for more details

Data Point	Data Type	Description
Equipment /State		
Equipment Original State Value	Int4	The original value of equipment state tag collector before it is updated by using MES Value Editor component or scripting
Equipment State Name	String	Current state name
Equipment State Path	String	Production model equipment path for equipment that is responsible for causing line downtime
Equipment State Split	Boolean	True is current downtime event has been split into multiple downtime events
Equipment State Type	String	See Setting Up Equipment - State Type for more details
State Begin Time	DateTime	Start time of the current state
State Duration	Float8	Duration of current state in minutes
State End Time	DateTime	End time of the current state

Equipment Meantime Data Points

Data Point	Data Types	Description
Equipment /Meantime		

Data Point	Data Types	Description
Equipment MTBF	Float8	The Mean Time (minutes) Between Failure for the selected period
Equipment Meantime Metrics Enabled	Boolean	True if Equipment Meantime Metrics are enabled for the current equipment state

Equipment General Data Points

Data Point	Data Types	Description
Equipment /General		
Delta Time Stamp	Float8	Time gap (minutes) between the rows of data.
From Time Stamp	DateTime	Start time (minutes) of the current period
Shift	String	Name of the current shift as set by the Ignition Schedule Management component and defined for the current line or by the value passed in the equipment shift tag collector
Shift Day Text	String	Name of the current day
Shift Day of Month	Int4	Int value of the current month
Shift Day of Week	Int4	Int value of the current day of the week
Shift Day of Year	Int4	Int value of the current day of the year

Data Point	Data Types	Description
Shift ISO Week of Year	Int4	Int value of the ISO week of the year
Shift Month Text	String	Name of the current month
Shift Month of Year	Int4	Int value of the current month of the year
Shift Start Date	DateTime	Start time of the current shift
Shift Week of Month	Int4	Int value of the current week of the month
Shift Week of Year	Int4	Int value of the current week of the year
Shift Year	Int4	Int value of the current year
To Time Stamp	DateTime	Endtime (minutes) of the current period

Equipment OEE Data Points

Data Point	Data Type	Description
Equipment/OEE		
Elapsed Time	Float8	Elapsed Time of current operation
OEE	Float8	OEE value for selected period

Data Point	Data Type	Description
OEE General Count	Long	Any count value other than infeed, outfeed, reject and waste value for the selected time period
OEE Infeed Count	Long	Equipment infeed count value for the selected period
OEE Infeed Count Equipment Path	String	Infeed count tag collector path
OEE Outfeed Count	Long	Equipment outfeed count value for the selected period
OEE Outfeed Count Equipment Path	String	Outfeed count tag collector path
OEE Reject Count	Long	Equipment reject count value for the selected period
Planned Downtime	Float8	Planned Downtime duration (Double) for selected period
Runtime	Float8	Planned Downtime duration (Double) for selected period
Short Stop Time	Float8	Short stop duration (Double) for selected period
Standard Rate	Float8	See standard rate for more details
Target Changeover Time	Float8	Amount of time in minutes set for Target Changeover. See Changeover Duration for more details
Unplanned Downtime	Float8	Unplanned Downtime duration (Double) for selected period

Data Point	Data Type	Description
Equipment/OEE /Availability		
Is Short Stop	Boolean	True if current equipment state is consider a shortstop. See Short Stop Threshold for more details
OEE Availability	Float8	OEE Availability value for selected period
Equipment/OEE /Performance		
OEE Performance	Float8	OEE Performance value for selected period
Infeed Standard Count	Float8	Calculated expected infeed based on standard rate
Equipment/OEE /Quality		
OEE Quality	Float8	OEE Quality value for selected period

Setting Values

The analysis results that are returned can be modified through the use of settings. Setting values provide a number of keywords as listed below.

Format for entering the keywords is k*eyword1=True, keyword2=100.0, keyword3=10.*

> ⚠ Settings like Enable Totalized Mode, Include Future, Last Values and Rollup Time span is meant for analysis selector and not for live analysis

Setting	Description	Use	Example
Date Format		All	

Setting	Description	Use	Example
	Date format fields can be customized with this setting e.g. 'YYYY/MM/dd hh:mm:ss a'		Date Format = `2017/04/12 19:45:30`
Enable Totalized Mode	This setting accumulates the count. Useful for charts where you wish to display the accumulated production count over time	Not valid for Live Analysis	Enable Totalized Mode = True
Include Future	Allows for count values to be calculated in the future. Useful for charts where you want to display target counts for future runs	Not valid for Live Analysis	Include Future = True
Last Values	Only the latest values are shown.	Not valid for Live Analysis	Last Values = True
OEE Availability Cap	The maximum value calculated can be capped with this setting	All	OEE Availability Cap = 100.0
OEE Performance Cap	The maximum value calculated can be capped with this setting	All	OEE Performance Cap = 100.0
OEE Quality Cap	The maximum value calculated can be capped with this setting	All	OEE Quality Cap = 100.0
Rollup Time Span	If the time (seconds) between downtime events is less than the rollup time and it is the same equipment and reason,	Not valid for Live Analysis	Rollup Time Span = 30

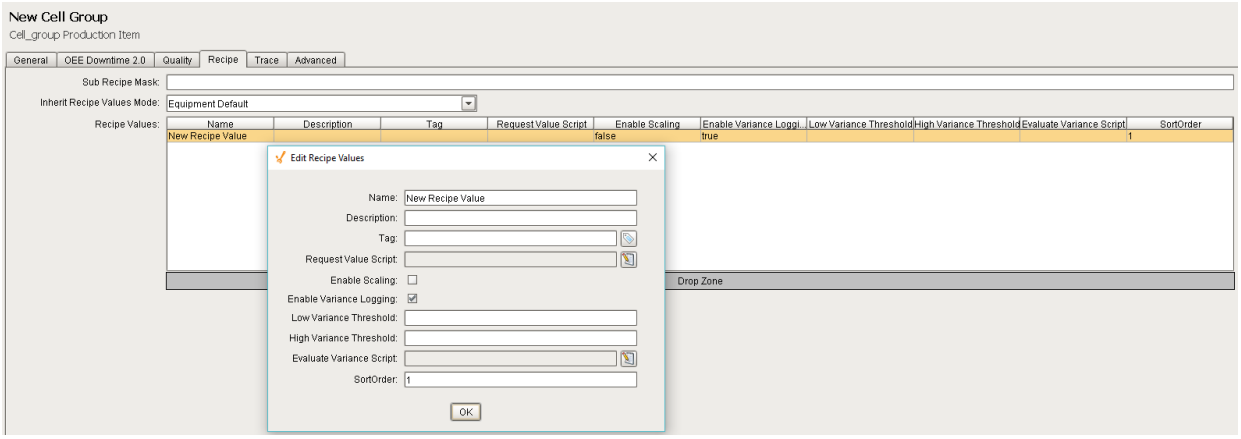
Setting	Description	Use	Example
	then it will rollup the event into one row in the results and will increase the occurrence count.		
Row Limit	The analysis can be limited to a certain number of rows.	All	Row Limit = 10

7.9.4 Recipe Tab

Recipe Values names can be inherited from the Enterprise, Site or Area level settings, or defined at the Line level.

At this level, it is possible to add configuration information regarding the tag, variance thresholds and the scripts used to evaluate them.

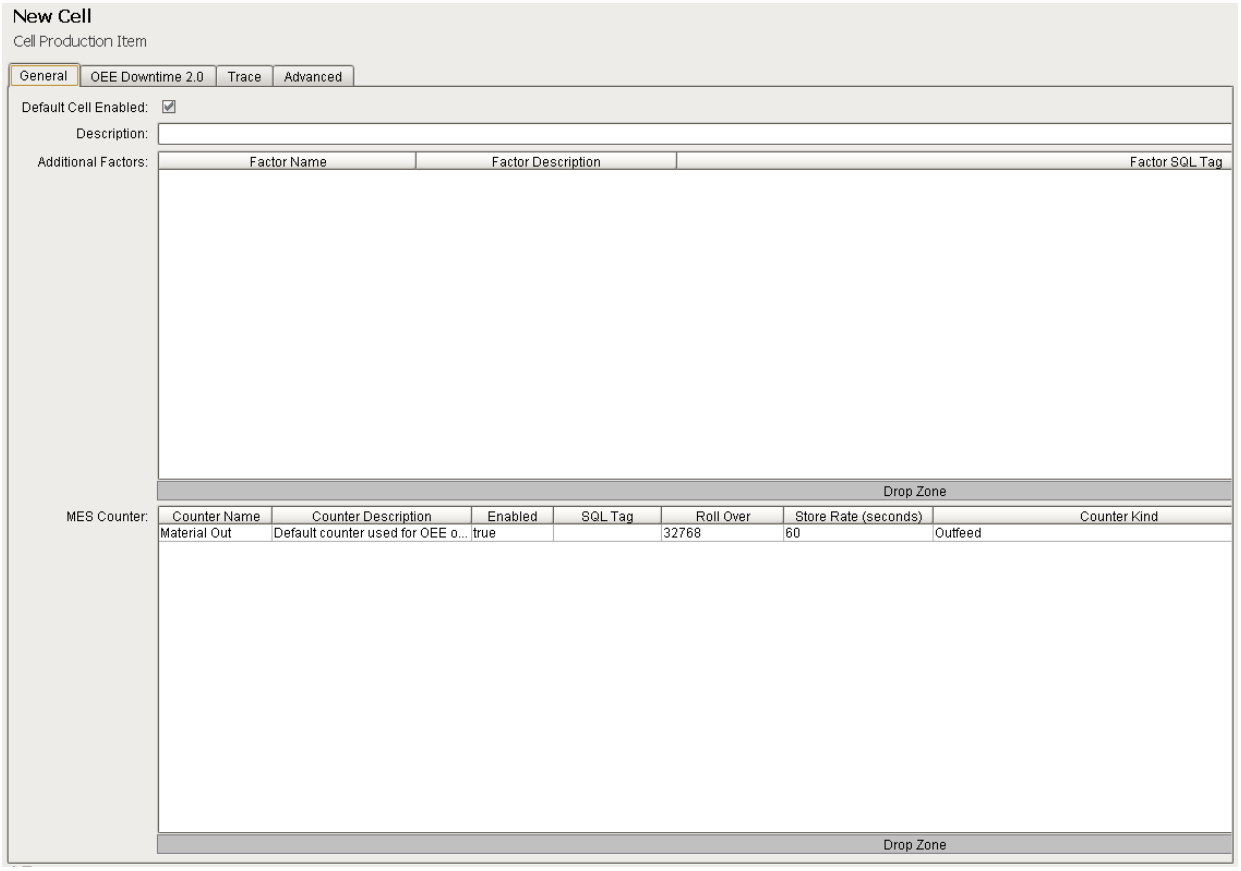
For more information please refer to the Recipe Production Model Configuration section.



Recipe Tab

7.10 Cell Settings

The Cell Production Item provides a configuration page for setting up a Production Cell for all the MES modules. The General and Advanced Tab are always present whereas each module when installed provides an additional tab.



7.10.1 General Tab - Additional Factors

The OEE Module collects and logs a number of downtime and production data values. However, what if other values outside of downtime and production values are of interest? Additional factors are the solution. Additional Factors are user defined data points that are logged along with the production and downtime information. Once they are logged, they can be shown in charts, tables and reports. Additionally, other analysis can be done by filtering and/or setting up comparisons by their values.

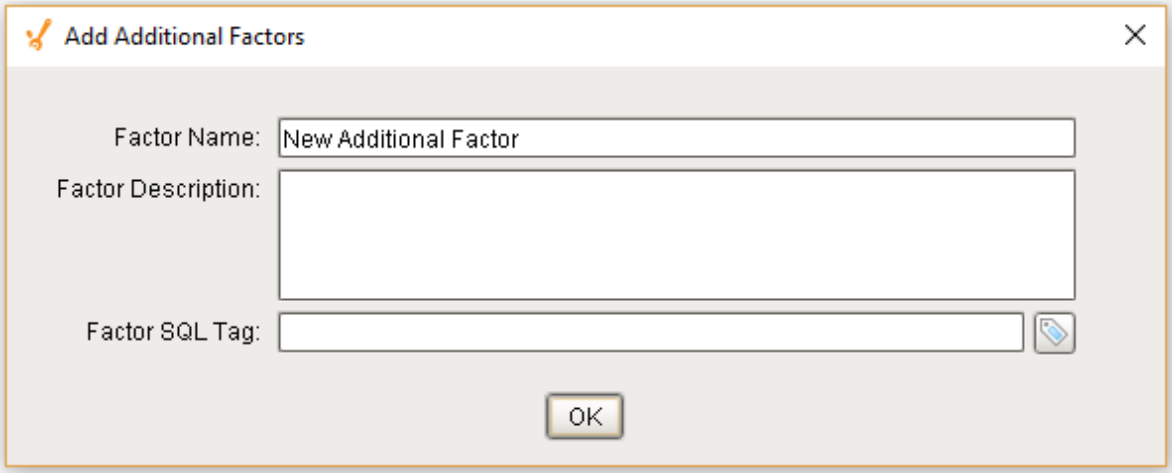
Additional Factors can added to the Line, Cell Group and Cell Production Items in the Production Model Designer.

Any value that can be read from an Ignition SQLTag can be added as an additional factor. This includes values derived from scripts, or from barcode readers, databases, calculations, PLCs, etc.

Any tag can be added as an additional factor. To configure, select a Line in the production model and select the **General** tab on the right. Right click on the **Additional Factors** table and select **New**. An additional factor is simply just a name and a tag.

Properties

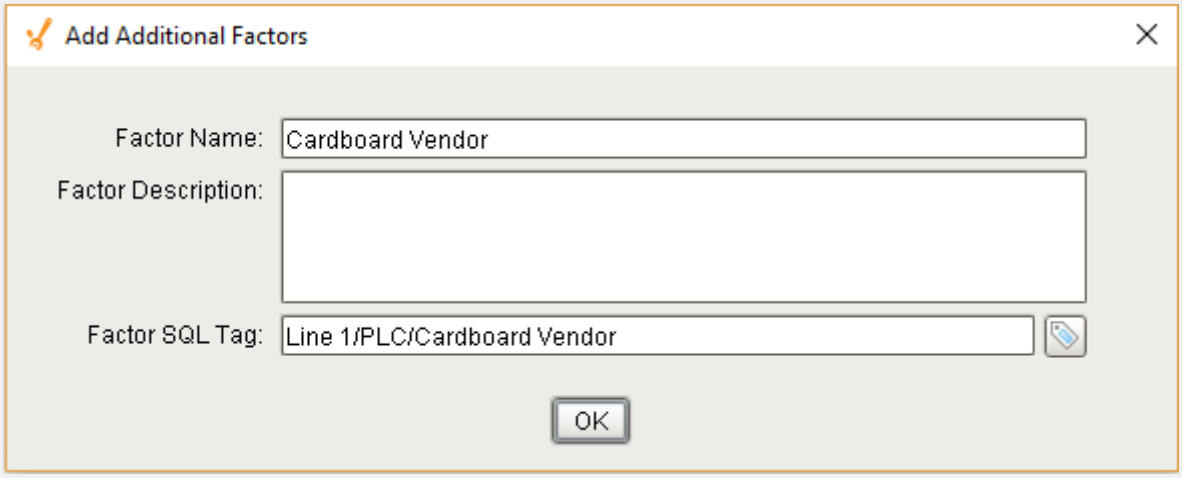
Factor Description	Optionally, this property can be set to a description for the additional factor. It is not used by the OEE Downtime and Scheduling Module other than for reference.	String
Factor Name	This reflects the name of additional factor that is configured in the designer.	String Read Only
Factor SQLTag	This reflects the Factor SQLTag setting that additional factor is configured for in the designer. It is the name of SQLTag to read the factor value from.	String Read Only

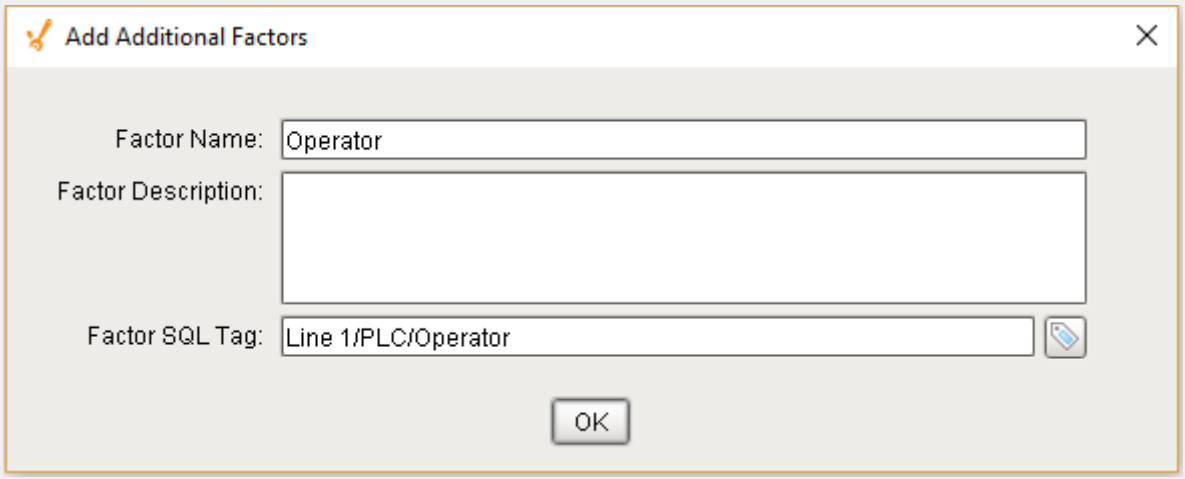


Example

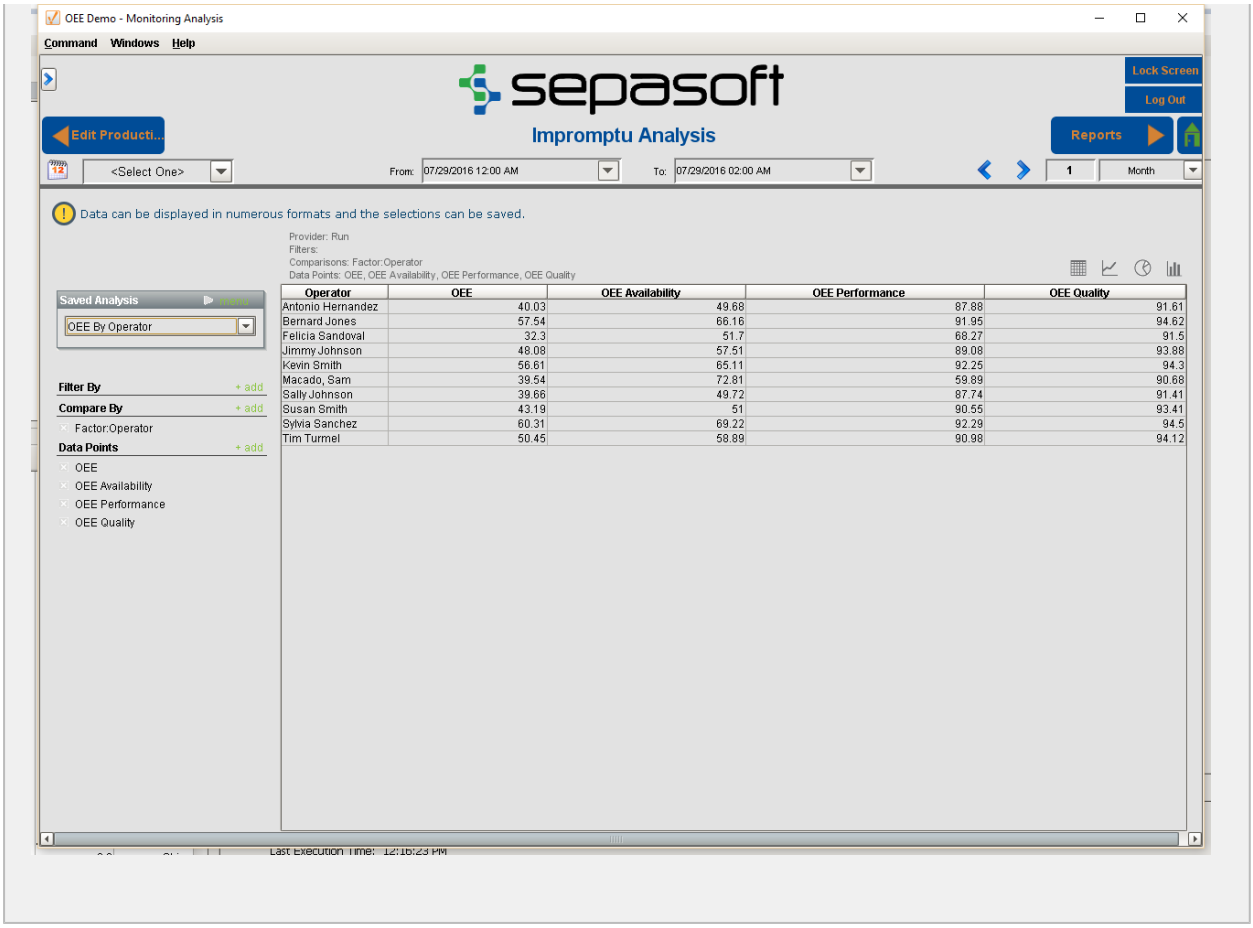
In the example, we have two factors, **Cardboard Vendor** and **Operator**. The operator can select the vendor that provided the cardboard or it can be obtained from some other source. Now, OEE and downtime results can be shown for each cardboard manufacturer. This can identify quality problems with raw material that directly affect production efficiency. With the operator setup as an additional factor, the operator's name will be

logged along with the production and downtime data. By doing so, OEE and downtime information can be filtered and grouped by the operator name. But this could just as well be the production crew, supervisor, maintenance crew or any other user defined value that can be monitored or entered into the system.





Adding these factors to a production line will allow us to capture the value of these tags whenever they change. In the impromptu analysis, we can then compare OEE values by our additional factor: Operator



7.10.2 General Tab - MES Counters

The MES Counters are used to associate Process Segments (Operations) with production counts. MES Counters record production counts 7/24, independent of scheduled production runs.

Counter names and the associated tag are defined in the Production Model. In the MES Management screen, the Quantity Source of Infeed and Material Process Segments can be set to use these MES counters.

MES counters are available for the Line, Cell Group, Cell or a Storage Unit production items in the Production Model Designer.

It is recommended to set up MES Counters at the cell level in order to accurately capture counts while indexing product on the line. Addition configuration must be accomplished with the cell settings in the production model.

The quantity from the MES counters can be obtained through scripting, see system.mes.getCountValue.

Adding MES Counter

To configure an MES Counter, select a Line, Cell Group, Cell or a Storage Unit in the production model and select the **General** tab on the right. Right click on the **MES Counter** table and select **New**. Properties can be set through the **Add MES Counter** Window.

Counter Name

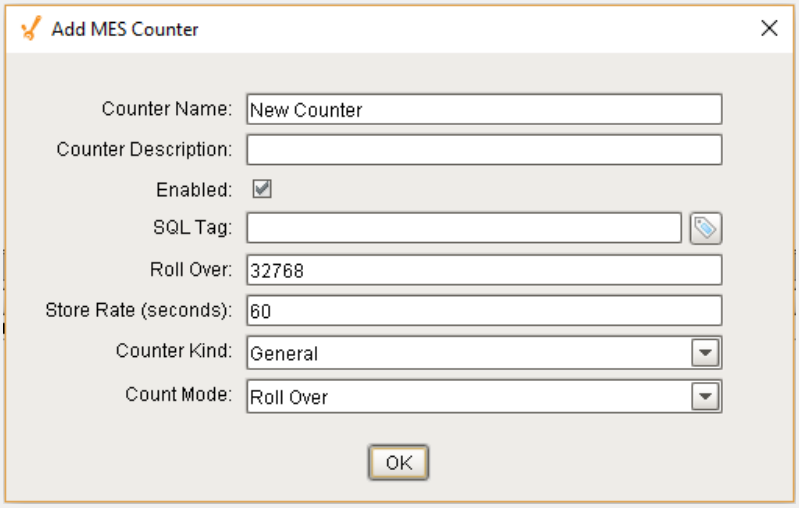
Name of the counter.

Counter Description

The description for the MES counter. This setting is not mandatory.

Enabled

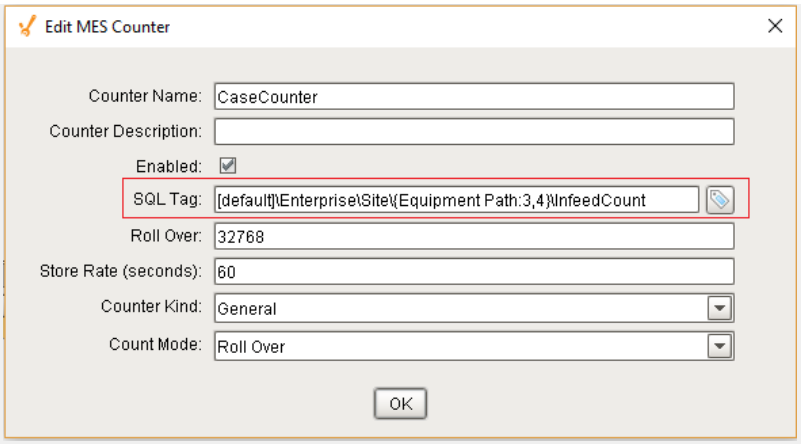
The counter can be enabled or disabled here.



SQL Tag

The path to the Tag Provider and ignition tag where the count value will come from is assigned to the MES counter here.

Parameterized Tag Paths can be used here which allows for indirection and exported MES Counters to be easily deployed to other equipment.



Roll Over

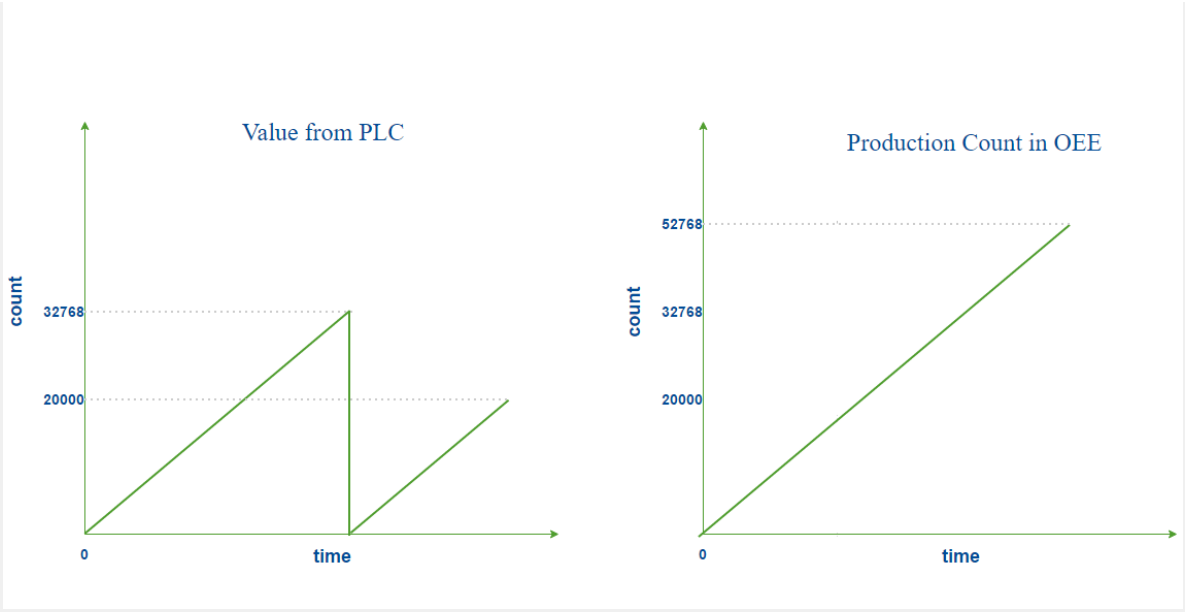
For PLC count tags that that do not get reset, they will eventually reach a finite maximum value at which point, the value will 'rollover' back to zero. The Roll Over setting allows you to define the value that should be added to the count tag whenever a roll over occurs. By default it is 32768 which equates to a 16 bit signed data value. Your setting will be dependent upon the datatype of your plc count tag.

During a production run, the incoming count value is added to the Roll Over setting multiplied by the number of times a rollover has occurred.

Example: production count value = incoming count value + 32768 * 3

Production Count	PLC Count tag	Rollover	Calculation
32700	32700	0	32700 + 32768 * 0
32750	32750	0	32700 + 32768 * 0
32918	150	1	150 + 32768 * 1
33068	300	1	300 + 32768 * 1

This value is only used when the **Count Mode** is set to **Rollover**.



Roll Over Count Mode

Store Rate

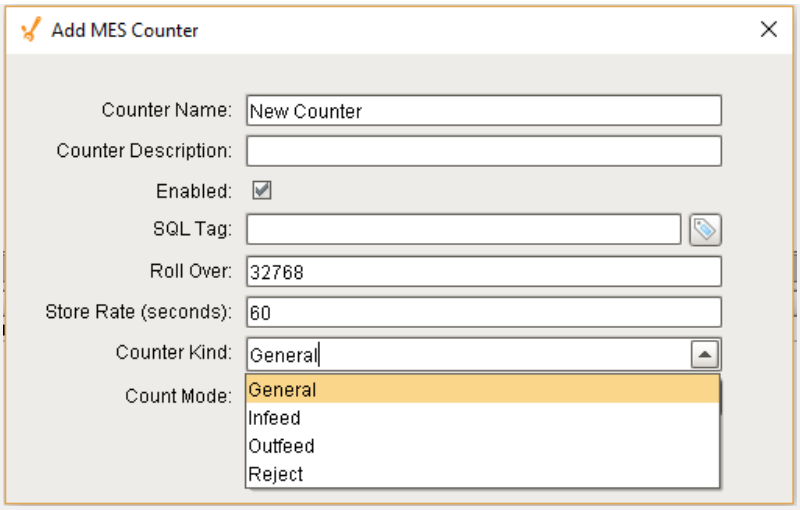
The MES counter will be captured and stored in the database after this specific interval in seconds if the value has changed If **Store Rate** is set to zero, every value change will be recorded.

Counter Kind

MES Counters can be set to four different kinds:

- Infeed
- Outfeed
- Reject
- General

Infeed, **Outfeed** and **Reject** kinds are used solely by the OEE module to determine which MES counter to use for OEE Performance, OEE Quality and production count information. The **General** kind can be used for any other count value.



Count Mode

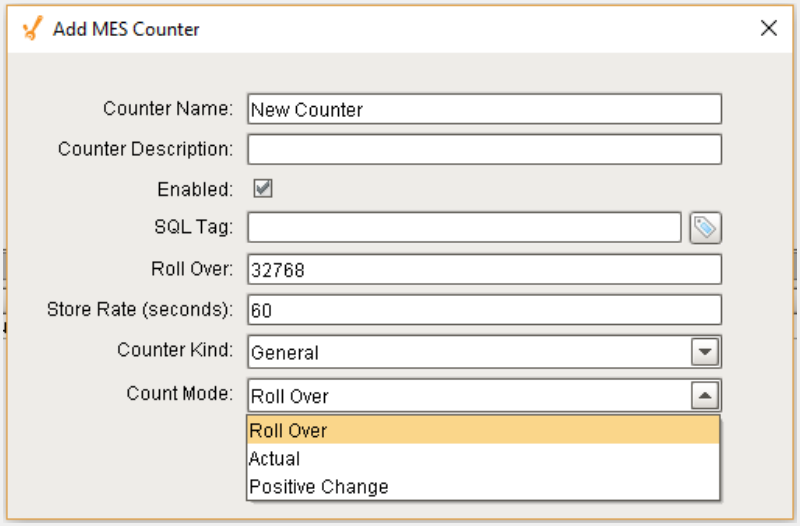
The Count Mode can be set to **Roll Over**, **Actual** and **Positive Change**.

Roll Over

See Rollover section for this count mode.

Actual

The Actual count mode simply uses whatever value is passed through the sql tag to represent the actual production counts. Production counts can go down as well as up.

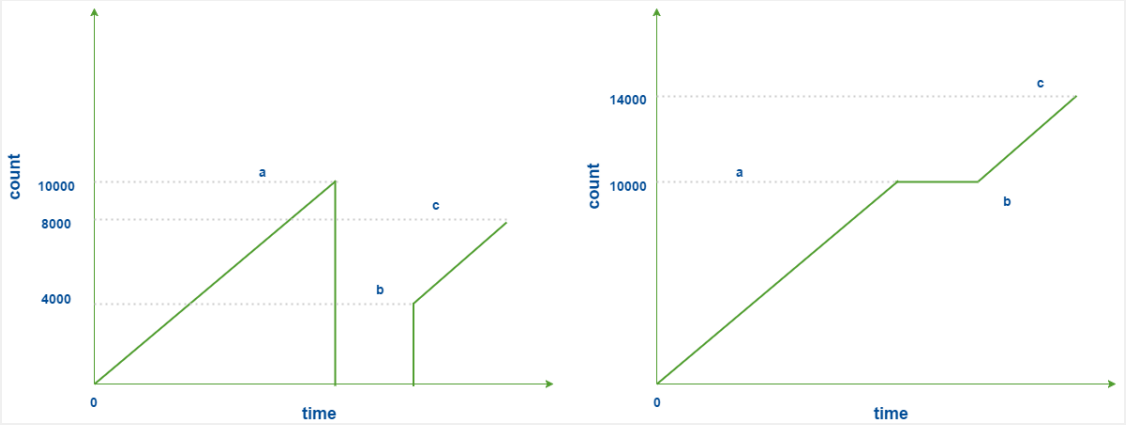


Positive Change

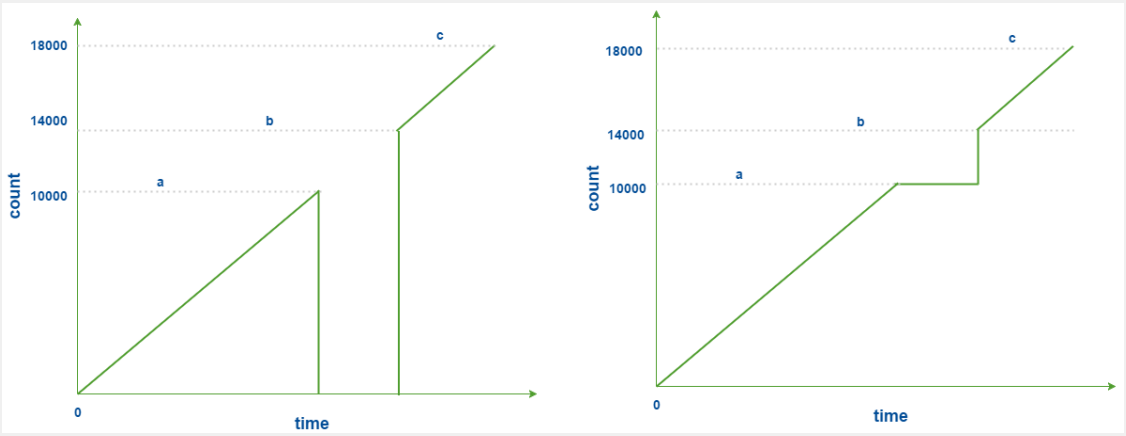
The Positive Change mode ignores any sql tag count values that are zero and will accumulate the counts. Three different cases are illustrated using the graphs shown.

PLC Count vs. OEE Count

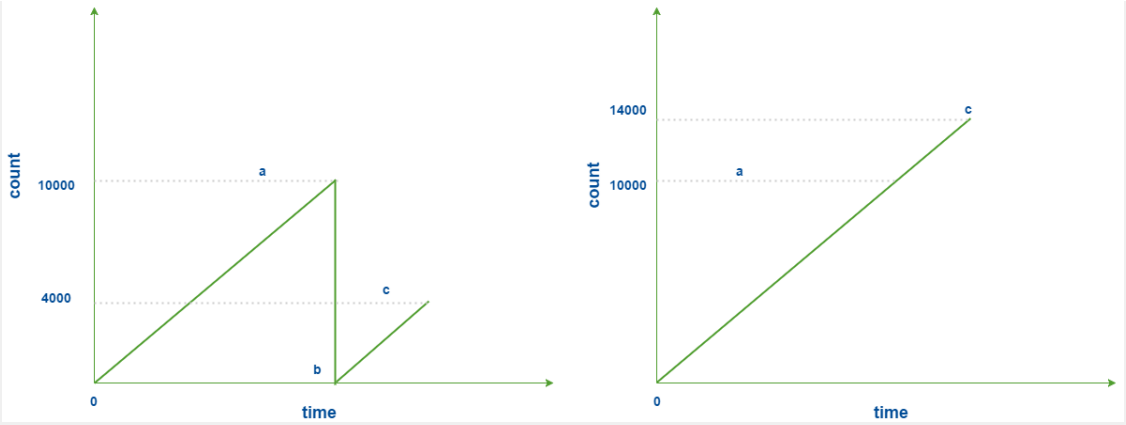
Case 1



Case 2



Case 3



Counter Rapid Development Features

Not only do counters allow for parameterization (as shown above), they also support copy, paste, import, and export features for rapid development. Configure the infeed counter for one Production Model Node (i.e. a Cell or Line) with parameterization, then copy and paste it to the other cells. Alternatively, copy and paste the Node itself and rename it for a similar effect.

7.10.3 OEE 2.0 Downtime Tab

The OEE Downtime 2.0 tab is specific to the OEE module and is available for the Line, Cell Group and Cell Production Items. A number of configuration settings are provided that can be used to obtain equipment mode, state and count values from ignition tags (whether plc tags, memory or expression tags).

Downtime Detection Mode

How line downtime is determined can be changed based on the selected Downtime Detection Mode. Valid options for the downtime detection mode are...

- Equipment State
- Key Reason (Cell Priority)
- Key reason (Neighbor Priority)
- Initial Cell

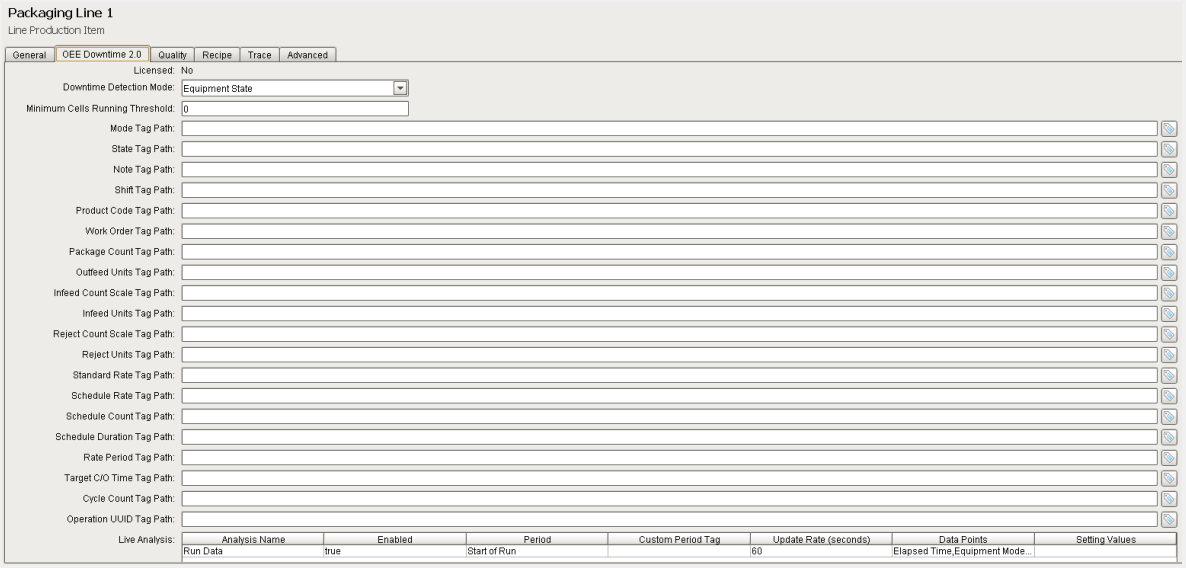
- Parallel Cells

Refer to Downtime Detection Mode for more information on the various Downtime Detection Methods.

> ⚠ Downtime Detection Mode is only available for the Line and Cell Group Production Item.

Minimum Cells Running Threshold

Minimum Cells Running Threshold determines how many cells in the Line (or Cell Group) must be running in order for the Line (or Cell Group) to be considered as Running.



Tag Collector Paths

Tag Collectors are provided to allow any of the parameters needed to drive OEE Metrics to be provided externally to the OEE module. Virtually all the Tag Collectors can be left blank in which case the OEE engine will determine the value from product code configuration information as defined in the OEE Material Manager or from internal calculations. Exceptions to this would be the equipment state and counts where needed.

> ⛔ When adding a tag to a Tag Collector, you must use memory tags. The Production Model will write values to any tags defined here as well as read the value whenever it changes from an external source.

Tag Collector	Tag Type	Data Type	Description

Mode	Memory	Integer	The Mode Tag, if provided, will be written to by the Production Model whenever the equipment mode changes based on Material Production Settings. The value of the mode tag can also be written to whenever the Mode value changes either indirectly from a plc tag (via tag change event) or from the HMI. The value of the Mode tag will be recorded for the current mode.
State	Memory	Integer	The State Tag path will generally come from a plc as the source of the current equipment state. Exceptions to this are at the Line level when using a downtime detection method other than Equipment State.
Downtime Note	Memory	String	Apart from scripting and Downtime table component, the downtime notes can be added by using tags.
Shift	Memory	String	When left blank, shifts defined in the Ignition Schedule Management component and defined in the Equipment Manager for a line will be used to determine the current shift. If a tag is provided here, whatever value is in the tag e.g. 'Shift A' will be recorded for the current shift.
Product Code	Memory	String	When left blank, the Product Code currently running on the line will be determined from the scheduled run as selected by the Scheduler or Run Director component. When a tag path is provided, the product code for the line or equipment (cell) will be determined from the value of the tag.
Work Order	Memory	String	When left blank, the Work Order currently running on the line will be determined from the scheduled run as selected by the Scheduler or Run Director component. When a tag path is provided, the product code for the line or equipment (cell) will be determined from the value of the tag.

Tag Collector Path	Tag Type	Data Type	Description
Package Count	Memory	Float	When left blank, the Package Count will be determined from the Package count setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Package Count for the line or equipment (cell) will be determined from the value of the tag. For more information on the Package Count, refer to the Material Production Settings section.
Line Outfeed Units	Memory	String	When left blank, the Line Outfeed Units will be determined from the Line Outfeed Units setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Line Outfeed Units for the line or equipment (cell) will be determined from the value of the tag. For more information on the Line Outfeed Units, refer to the Material Production Settings section.
Infeed Count Scale	Memory	Float	When left blank, the Infeed Count Scale will be determined from the Infeed Count Scale setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Infeed Count Scale for the line or equipment (cell) will be determined from the value of the tag. For more information on the Infeed Count Scale, refer to the Material Production Settings section.
Line Infeed Units	Memory	String	When left blank, the Line Infeed Count Scale will be determined from the Infeed Count Scale setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Infeed Count Scale for the line or equipment (cell) will be determined from the value of the tag. For more information on the Line Infeed Units, refer to the Material Production Settings section.

Tag Collector Path	Tag Type	Data Type	Description
Reject Count Scale	Memory	Float	When left blank, the Reject Count Scale will be determined from the Reject Count Scale setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Reject Count Scale for the line or equipment (cell) will be determined from the value of the tag. For more information on the Reject Count Scale, refer to the Material Production Settings section.
Line Reject Units	Memory	String	When left blank, the Line Reject Units will be determined from the Line Reject Units setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Line Reject Units for the line or equipment (cell) will be determined from the value of the tag. For more information on the Line Reject Units, refer to the Material Production Settings section.
Standard Rate	Memory	Float	When left blank, the Standard Rate will be determined from the Standard Rate setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Standard Rate for the line or equipment (cell) will be determined from the value of the tag. For more information on the Standard Rate, refer to the Material Production Settings section.
Schedule Rate	Memory	Float	When left blank, the Schedule Rate will be determined from the Schedule Rate setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Schedule Rate for the line will be determined from the value of the tag. For more information on the Schedule Rate, refer to the Material Production Settings section.

Tag Collector Path	Tag Type	Data Type	Description
			Schedule Rate Tag Path is only available for the Line Production Item.
Schedule Count	Memory	Integer	When left blank, the Schedule Count will be determined from the scheduled run as selected by the Scheduler or Run Director component. When a tag path is provided, the Schedule Count for the line or equipment (cell) will be determined from the value of the tag. The Schedule Count provides the number of units scheduled to be produced. ⚠ Schedule Count Tag Path is only available for the Line Production Item.
Schedule Duration	Memory	Integer	When left blank, the Schedule Duration will be determined from the scheduled run as selected by the Scheduler or Run Director component. When a tag path is provided, the Schedule Duration for the line or equipment (cell) will be determined from the value of the tag. The Schedule Duration provides the expected runtime required for the number of units scheduled to be produced and is calculated by the Schedule Rate.
Rate Period	Memory	String	When left blank, the Rate Period will be determined from the Rate Period setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Rate Period for the line or equipment (cell) will be determined from the value of the tag. For more information on the Rate Period, refer to the Material Production Settings section.
Target C /O Time	Memory	Integer	

Tag Collector Path	Tag Type	Data Type	Description
			When left blank, the Target C/O (Changeover) Time will be determined from the Changeover settings for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Target C/O (Changeover) Time for the line or equipment (cell) will be determined from the value of the tag. For more information on the Target C/O (Changeover) Time, refer to the Material Production Settings section. ⚠ Target C/O Time Tag Path is only available for the Line Production Item.
Cycle Count	Memory	Float	When left blank, the Cycle Count will be determined by the OEE Module. When a tag path is provided, the Cycle Count for the line or equipment (cell) will be determined from the value of the tag.
Operation UUID	Memory	String	When left blank, the Operation UUID will be determined from the currently running Operation on the Line or equipment (cell). When a tag path is provided, the Operation UUID for the line or equipment (cell) can be determined from the value of the tag. The purpose for this tag is to be able to provide OEE analysis data when production runs are not scheduled or started using the Run Director or Schedule Selector components, or scripting functions. In this case a tag can be used to provide a Run Identifier value i.e. Run_4253_XX. The Analysis Selector provides the ability to pull the Operation UUID as part of analysis, whether it is an internally generated Operation UUID or a passed Run Identifier.

Tag Collector Paths can be parameterized with {Equipment Path} to utilize indirection and more rapidly implement the production model. See Parameterized Tag Paths for more details.

Live Analysis

Live Analysis provides a flexible way of customizing your application to provide a set of real-time tag values that can be accessed from the Ignition designer and used in your application to provide real-time production monitoring. Live Analysis is configured in the OEE 2.0 Downtime tab of the Production Model Designer for the Line, Cell Group and Cell production items. When a Live Analysis is created, a corresponding set of tags is created in the MES Tag Provider that provide the real-time status of those datapoints based upon the Period defined for the Live Analysis. You can create multiple Live Analysis and use those tags to drive HMI displays.

To create a new Live Analysis:

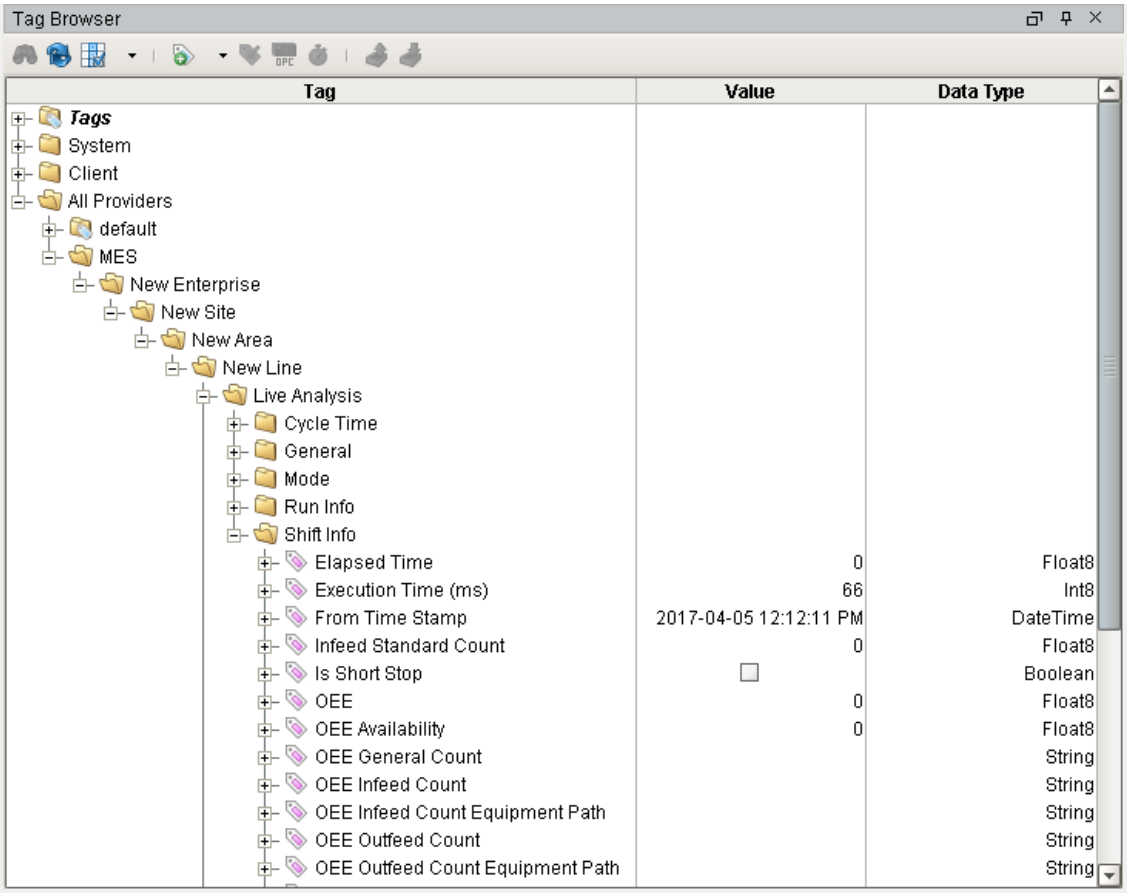
- Right click on the Live Analysis panel on the OEE 2.0 Downtime Tab in the Production Model Designer.
- Provide a Name
- Select the Period that the Live Analysis datapoints will return a value for. Valid options are Shift, Day (Midnight), Day (Production), Start of Run, Top of Hour, Custom Period Tag
- Select the frequency for how often the tag values will be updated. Default value is 60 seconds. Minimum value is 10 seconds
- Select the desired Data Points
- Add any further Settings Values required

⚠ You cannot select all Data Points in one Live Analysis. The maximum length string for Data Points is 1024 characters

Live Analysis:

Analysis Name	Enabled	Period	Custom Period Tag	Update Rate (seconds)	Data Points	Setting Values
Cycle Time	true	Shift		60	Average Normal Cycl...	
General	true	Shift		60	Delta Time Stamp,Eq...	
Mode	true	Shift		60	Equipment Mode Na...	
Run Info	true	Start of Run		60	Equipment Cell Orde...	
Shift Info	true	Shift		60	Elapsed Time,Infeed ...	

Live Analysis Settings Panel in the OEE 2.0 Downtime Tab



MES Tag Provider Live Analysis Tags

Live Analysis Settings

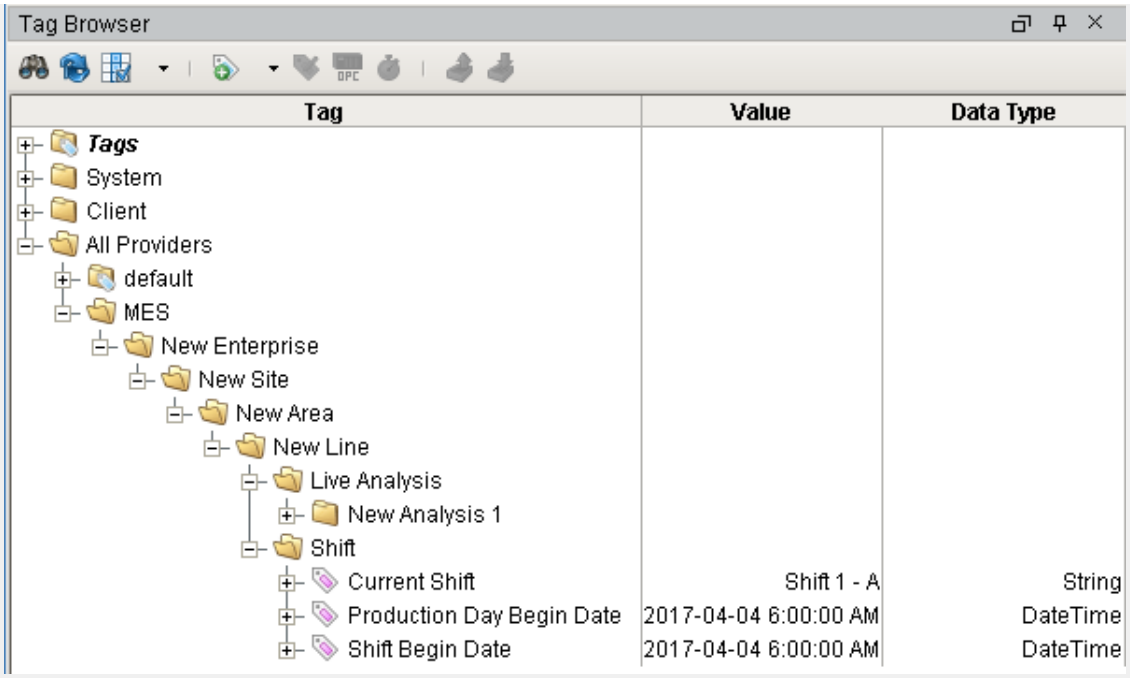
Setting	Description
Analysis Name	The name for the live analysis.
Enabled	The live analysis can be enabled or disabled with this setting.
Period	The duration of analysis can be set by **Shift**, **Day (midnight)**, **Day (production)**, **Start of Run**, **Top of Hour** or **Custom Period Tag**.
Custom Period Tag	A tag can be assigned to define the start datetime for a custom period. The end time will be the current time. It takes value in the date time data type. Example for a valid value for the custom period tag is: **`2017/04/04 14:00:00`**

Setting	Description
Update Rate	The rate in seconds by which the live analysis is updated. The minimum update rate is 60 seconds.
Data Points	Data points allows you to pick and choose the values you wish to access through tags. See the table below for the listing of available data points.

Shift Data Points

When creating a Live Analysis, the following shift data points will be automatically created.

Data Point	Data Type	Description
Current Shift	String	The currently running shift as defined in the Ignition Schedule Management component or passed from the Shift Tag Collector path
Production Day Begin Date	DateTime	Production start time
Shift Begin Date	DateTime	Start time of the current shift



Analysis Data Points and Settings are used by Live Analysis, the MES Analysis Selector and MES Analysis Controller components, the MES Analysis Data Source for reporting and the MES Analysis Settings object.

Equipment Data Points

Data Point	Data Type	Description
Equipment		
Equipment Cell Order	Int4	Integer value that determines the cell order of the equipment within the line. Is set to *null* for the line. Is set to 0 for first cell within each cell group
Equipment Name	String	Name of the equipment as defined in the production model
Equipment Note	String	Any note that has been recorded for this piece of equipment through the Note tag collector path in the Production model will be exposed here.
	DateTime	

Data Point	Data Type	Description
Equipment Operation Begin		Start Date time of the currently running operation on this equipment
Equipment Operation Sequence	Int4	The ordinal number (integer) of operation
Equipment Path	String	Production model path for this equipment
Equipment Type	String	Can be Line, Cell Group or Cell
Execution Time (ms)	Int8	Time taken to execute and update the Live Analysis. Used mainly for performance debugging
From Time Stamp	DateTime	Start Date Time of current data point results
Infeed Units	String	See Infeed Units for more details
Is Key Cell	Boolean	See Key Reason for more details
Operation UUID	String	Unique Identifier for currently running operation
Outfeed Units	String	See Outfeed Units for more details
Product Code	String	Product code currently being processed on this equipment
Rate Period	String	See Rate Period for more details

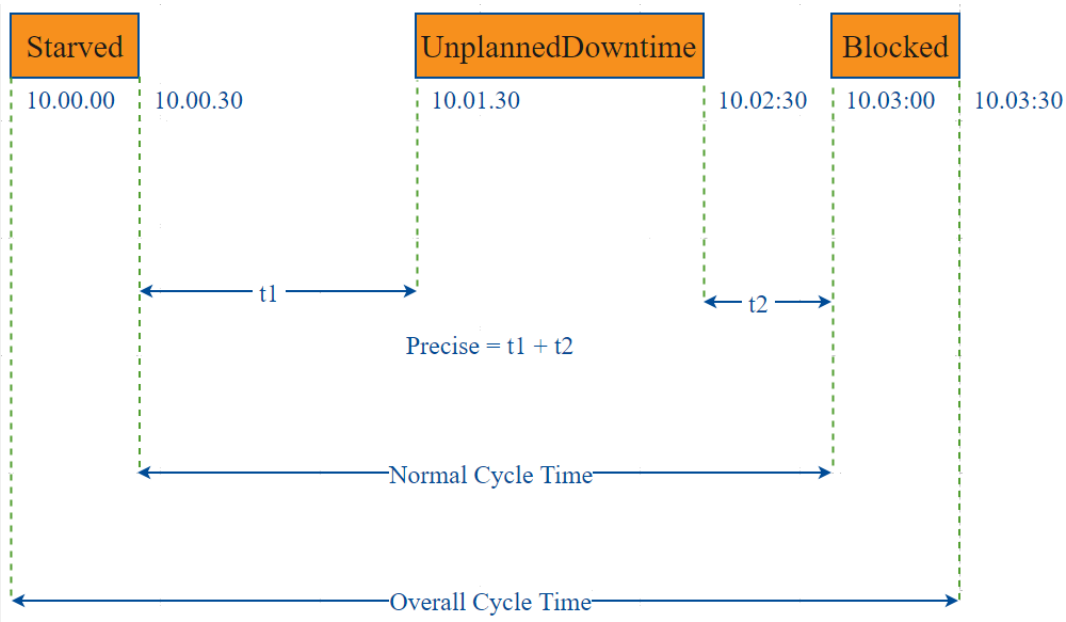
Data Point	Data Type	Description
Reject Units	String	See Reject Units for more details
To Time Stamp	DateTime	End Date Time of current data point results
Work Order	String	Work order currently being processed on this equipment

Equipment Count Data Points

Data Point	Data Type	Description
Equipment\Count		*Any defined counters for the production item will also appear in this folder
Equipment Infeed Scale	Float8	See Infeed Count Scale for more details
Equipment Package Count	Float8	See Package Count for more details
Equipment Reject Scale	Float8	See Reject Count Scale for more details
Outfeed-Material Out	String	Value of the default MES Counter used for OEE outfeed count

Equipment Cycle Time Data Points

The Cycle Time data points provides a number of metrics that can be used to measure the amount of time required to produce one piece. It is often used to gain an understanding of variations in production. Live Analysis provides Target, Normal, Overall and Precise Cycle Time metrics.



Data Point	Data Type	Description
Equipment\Cycle Time		
Relative Cycle Count	String	Relative Cycle Count is how many occurred for the compare by.
Target Cycle Time	Float8	Also known as Takt time, it is how often a piece must be produced to meet customer demand. It is often used to pace a production line, and it is a calculated number in seconds.
Total Cycle Count	String	Total Cycle Count is accumulative, it is sum total of all the cycle count.
Equipment\Cycle Time\Normal		**Normal Cycle Time is the actual cycle ignoring the equipment states like starved, blocked, etc.**
Average Normal Cycle Time	Float8	Average Normal cycle time in seconds for the time period selected
	Float8	

Data Point	Data Type	Description
Max Normal Cycle Time		Max Normal cycle time in seconds for the time period selected
Min Normal Cycle Time	Float8	Min Normal cycle time in seconds for the time period selected
Normal Cycle Time	Float8	Normal Cycle Time in seconds is the actual cycle ignoring the equipment states like starved, blocked, etc.
Equipment\Cycle Time\Overall		**Overall Cycle Time is the cycle including states like downtime, starved, blocked, etc.**
Average Overall Cycle Time	Float8	Average Overall cycle time in seconds for the time period selected
Max Overall Cycle Time	Float8	Max Overall cycle time in seconds for the time period selected
Min Overall Cycle Time	Float8	Min Overall cycle time in seconds for the time period selected
Overall Cycle Time	Float8	Overall Cycle Time in seconds is the cycle including states like downtime, starved, blocked, etc.
Equipment\Cycle Time\Precise		**Precise Cycle Time is the cycle time ignoring all the equipment states**
Average Precise Cycle Time	Float8	Average Precise cycle time in seconds for the time period selected
Max Precise Cycle Time	Float8	Max Precise cycle time in seconds for the time period selected
	Float8	

Data Point	Data Type	Description
Min Precise Cycle Time		Min Precise cycle time in seconds for the time period selected
Precise Cycle Time	Float8	Precise cycle time in seconds excluding states like planned downtime, unplanned downtime, starved and blocked.

Line Data Points

The Line Data points returns data for the line regardless of the Equipment the Live Analysis has been set up for.

Data Point	Data Type	Description
Line /Downtime		
Line Downtime Equipment Name	String	Name of the equipment that is responsible for causing line downtime
Line Downtime Equipment Path	String	Production model equipment path for equipment that is responsible for causing line downtime
Line Downtime Event Sequence	Int4	Every downtime event on the line is provided with an incrementing sequence number
Line Downtime Note	String	Note entered at the Line level

Data Point	Data Type	Description
Line Downtime Occurrence Count	Int4	Number of downtime events for the selected period.
Line Downtime Reason	String	The line or cell group (sub line) downtime reason. 1. When the line is down the Line Downtime Reason is the same as the Line State Name. 2. When the line is up the Line Downtime Reason is blank.
Line Downtime Reason Path	String	The full reason name for line or cell group (sub line) downtime reason. Line State name including State Class i.e. *Default/Cell Faulted*
Line Downtime Reason Split	Boolean	The line downtime reason split indicator. True is current downtime event has been split into multiple downtime events
Line Downtime State Time Stamp	DateTime	The time stamp for the equipment state change of the cell group (sub line) or cell that caused the line down time even.
Line /Meantime		
Line MTBF	Float8	The calculated Meantime (minutes) Between Failure for the selected period. Refer to Setting Up Equipment States - Meantime Metrics for more details.

Data Point	Data Type	Description
Line Meantime Metrics Enabled	Boolean	Returns if Meantime metrics have been enabled for this equipment.
Line /Schedule		
Line Schedule Available	Boolean	True if this operation was scheduled
Line Schedule Available Time	Float8	Time in minutes for available production time adjusted for line schedule availability and mode.
Line Standard Count	String	Amount of product that should have been produced based on the line schedule available time and line standard rate
Line Standard Count Variance	String	Variance between standard count and actual count
Line Target Count	String	Amount of product that should have been produced based on the line schedule available time and line schedule rate
Line Target Count Variance	String	Variance between line scheduled count and line OEE outfeed count.
Schedule Rate	Float8	See Schedule Rate for more details

Data Point	Data Type	Description
Line/State		
Line State Duration	Float8	The line or cell group (sub line) downtime event duration in minutes.
Line State Event Begin	DateTime	The line or cell group (sub line) downtime event begin date time.
Line State Event End	DateTime	The line or cell group (sub line) downtime event end date time.
Line State Event Sequence	Int4	The equipment state event sequence number.
Line State Name	String	The line or cell group (sub line) state. 1. When the line is down the Line Downtime Reason is the same as the state name. 2. When the line is up the Line Downtime Reason is blank.
Line State Override Scope	String	The state override scope for a line or cell group (sub line). See Setting Up Equipment - Override Scope for more details
Line State Override Type	String	The state override type for a line or cell group (sub line). See Setting Up Equipment - Override for more details
Line State Type	String	The line or cell group (sub line) state type. See Setting Up Equipment - State Type for more details

Data Point	Data Type	Description
Line State Value	Int4	The line or cell group (sub line) downtime state code. See Setting Up Equipment - State Code for more details

Equipment Mode & State Data Points

Data Point	Data Type	Description
Equipment /Mode		
Equipment Mode Name	String	Name of the current mode. See Setting Up Equipment Modes for more details
Equipment Mode Type	String	Name of the current mode type. See Setting Up Equipment Modes for more details
Equipment Mode Value	Int4	Name of the current mode. See Setting Up Equipment Modes for more details
Mode Begin Time	DateTime	Start time of the current mode
Mode Duration	Float8	Duration of the current mode in minutes
Mode End Time	DateTime	End time of the current mode
OEE Enabled	Boolean	See Setting Up Equipment Modes - OEE Enabled for more details
Production Counts Enabled	Boolean	See Setting Up Equipment Modes - OEE Enabled for more details

Data Point	Data Type	Description
Equipment /State		
Equipment Original State Value	Int4	The original value of equipment state tag collector before it is updated by using MES Value Editor component or scripting
Equipment State Name	String	Current state name
Equipment State Path	String	Production model equipment path for equipment that is responsible for causing line downtime
Equipment State Split	Boolean	True is current downtime event has been split into multiple downtime events
Equipment State Type	String	See Setting Up Equipment - State Type for more details
State Begin Time	DateTime	Start time of the current state
State Duration	Float8	Duration of current state in minutes
State End Time	DateTime	End time of the current state

Equipment Meantime Data Points

Data Point	Data Types	Description
Equipment /Meantime		

Data Point	Data Types	Description
Equipment MTBF	Float8	The Mean Time (minutes) Between Failure for the selected period
Equipment Meantime Metrics Enabled	Boolean	True if Equipment Meantime Metrics are enabled for the current equipment state

Equipment General Data Points

Data Point	Data Types	Description
Equipment /General		
Delta Time Stamp	Float8	Time gap (minutes) between the rows of data.
From Time Stamp	DateTime	Start time (minutes) of the current period
Shift	String	Name of the current shift as set by the Ignition Schedule Management component and defined for the current line or by the value passed in the equipment shift tag collector
Shift Day Text	String	Name of the current day
Shift Day of Month	Int4	Int value of the current month
Shift Day of Week	Int4	Int value of the current day of the week
Shift Day of Year	Int4	Int value of the current day of the year

Data Point	Data Types	Description
Shift ISO Week of Year	Int4	Int value of the ISO week of the year
Shift Month Text	String	Name of the current month
Shift Month of Year	Int4	Int value of the current month of the year
Shift Start Date	DateTime	Start time of the current shift
Shift Week of Month	Int4	Int value of the current week of the month
Shift Week of Year	Int4	Int value of the current week of the year
Shift Year	Int4	Int value of the current year
To Time Stamp	DateTime	Endtime (minutes) of the current period

Equipment OEE Data Points

Data Point	Data Type	Description
Equipment/OEE		
Elapsed Time	Float8	Elapsed Time of current operation
OEE	Float8	OEE value for selected period

Data Point	Data Type	Description
OEE General Count	Long	Any count value other than infeed, outfeed, reject and waste value for the selected time period
OEE Infeed Count	Long	Equipment infeed count value for the selected period
OEE Infeed Count Equipment Path	String	Infeed count tag collector path
OEE Outfeed Count	Long	Equipment outfeed count value for the selected period
OEE Outfeed Count Equipment Path	String	Outfeed count tag collector path
OEE Reject Count	Long	Equipment reject count value for the selected period
Planned Downtime	Float8	Planned Downtime duration (Double) for selected period
Runtime	Float8	Planned Downtime duration (Double) for selected period
Short Stop Time	Float8	Short stop duration (Double) for selected period
Standard Rate	Float8	See standard rate for more details
Target Changeover Time	Float8	Amount of time in minutes set for Target Changeover. See Changeover Duration for more details
Unplanned Downtime	Float8	Unplanned Downtime duration (Double) for selected period

Data Point	Data Type	Description
Equipment/OEE /Availability		
Is Short Stop	Boolean	True if current equipment state is consider a shortstop. See Short Stop Threshold for more details
OEE Availability	Float8	OEE Availability value for selected period
Equipment/OEE /Performance		
OEE Performance	Float8	OEE Performance value for selected period
Infeed Standard Count	Float8	Calculated expected infeed based on standard rate
Equipment/OEE /Quality		
OEE Quality	Float8	OEE Quality value for selected period

Setting Values

The analysis results that are returned can be modified through the use of settings. Setting values provide a number of keywords as listed below.

Format for entering the keywords is k*eyword1=True, keyword2=100.0, keyword3=10.*

> ⚠ Settings like Enable Totalized Mode, Include Future, Last Values and Rollup Time span is meant for analysis selector and not for live analysis

Setting	Description	Use	Example
Date Format		All	

Setting	Description	Use	Example
	Date format fields can be customized with this setting e.g. 'YYYY/MM/dd hh:mm:ss a'		Date Format = `2017/04/12 19:45:30`
Enable Totalized Mode	This setting accumulates the count. Useful for charts where you wish to display the accumulated production count over time	Not valid for Live Analysis	Enable Totalized Mode = True
Include Future	Allows for count values to be calculated in the future. Useful for charts where you want to display target counts for future runs	Not valid for Live Analysis	Include Future = True
Last Values	Only the latest values are shown.	Not valid for Live Analysis	Last Values = True
OEE Availability Cap	The maximum value calculated can be capped with this setting	All	OEE Availability Cap = 100.0
OEE Performance Cap	The maximum value calculated can be capped with this setting	All	OEE Performance Cap = 100.0
OEE Quality Cap	The maximum value calculated can be capped with this setting	All	OEE Quality Cap = 100.0
Rollup Time Span	If the time (seconds) between downtime events is less than the rollup time and it is the same equipment and reason,	Not valid for Live Analysis	Rollup Time Span = 30

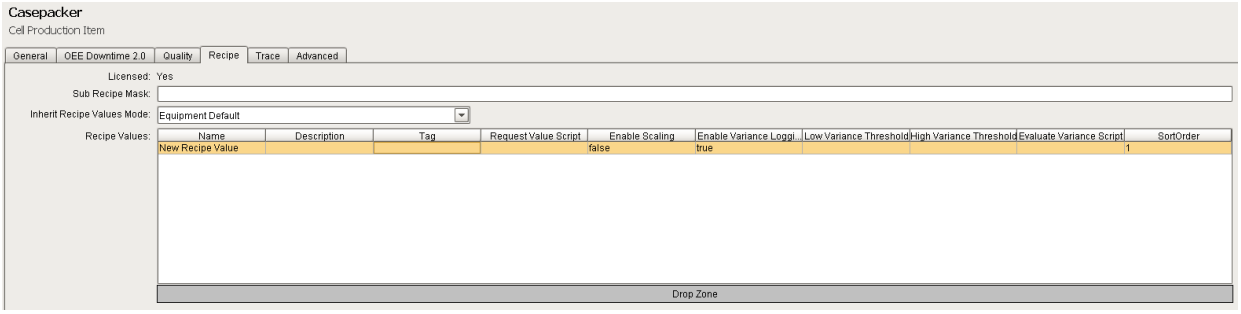
Setting	Description	Use	Example
	then it will rollup the event into one row in the results and will increase the occurrence count.		
Row Limit	The analysis can be limited to a certain number of rows.	All	Row Limit = 10

7.10.4 Recipe Tab

Recipe Values names can be inherited from the Enterprise, Site or Area level settings, or defined at the Line level.

At this level, it is possible to add configuration information regarding the tag, variance thresholds and the scripts used to evaluate them.

For more information please refer to the Recipe Production Model Configuration section.

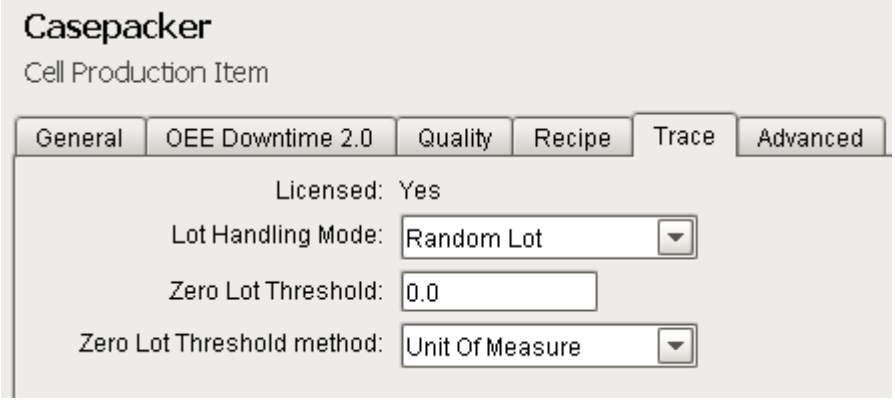


Recipe Tab

7.10.5 Trace Tab

The Trace tab allows you to define the **Lot Handling Mode** for the selected line.

Refer to the Lot Handling Mode section in the Track & Trace Module help.



Trace Tab

7.11 Location Settings

Locations are used exclusively by the SPC module. They define the locations that are associated with sample collection.

7.11.1 General Tab

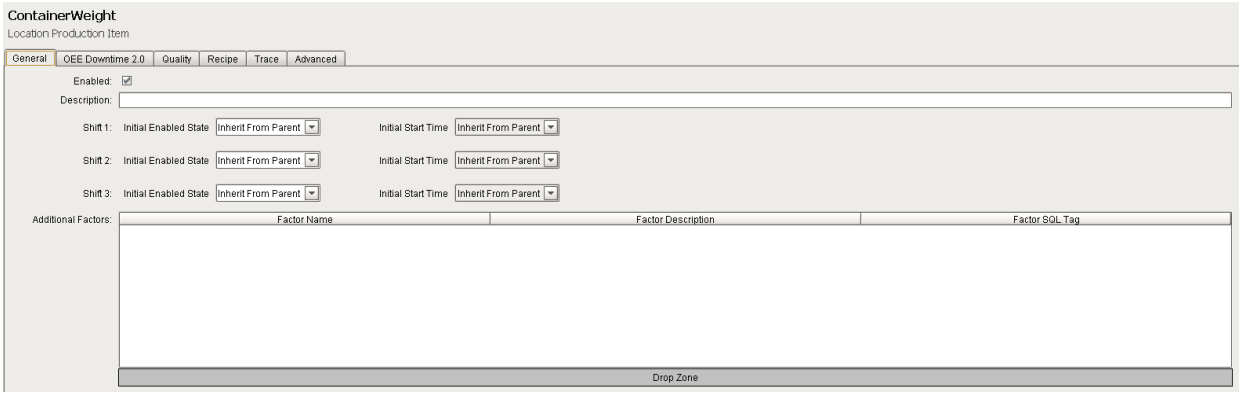
Shifts

Locations can be configured to inherit from the default shifts for the site, or they can be overridden in this tab.

Additional Factors

Additional Factors are user defined data points that are logged along with the sample data. It extends the SPC Module analysis by allowing you to view the additional factor values in charts, tables, and reports. Additionally, SPC analysis can be done by filtering and/or setting up comparisons by the additional factor values.

See SPC Additional Factors for more information on setting up Additional Factors.



General Tab

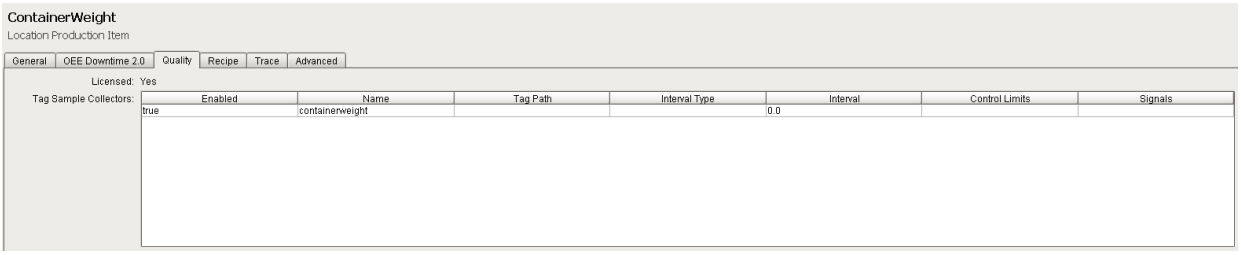
7.11.2 Quality Tab

The Quality tab is specific to the SPC module.

Tag Sample Collectors

You can define Tag Sample collectors in this table that will automate the collection of sample data.

Please refer to the Tag Sample Collectors section in the SPC Module help.



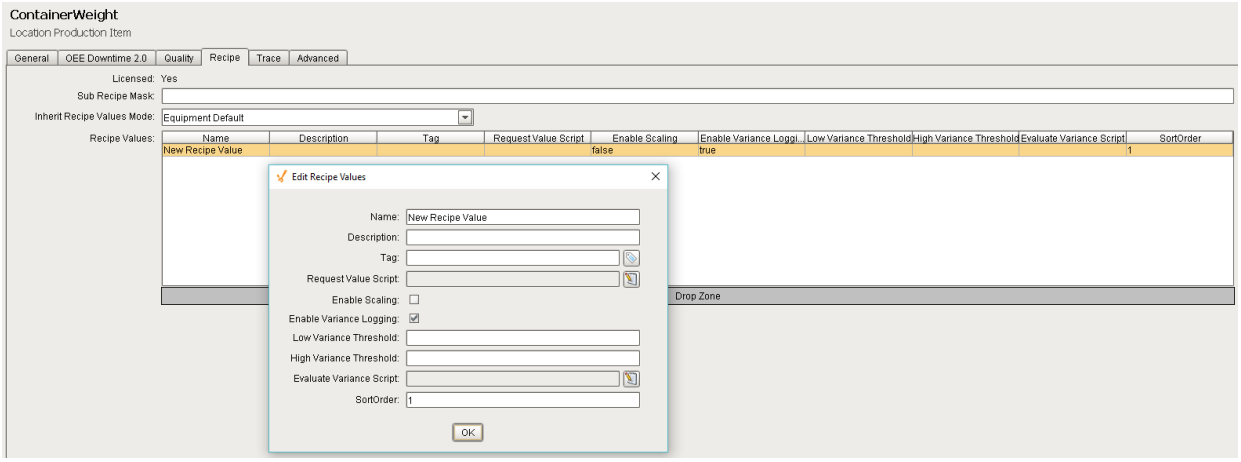
Quality Tab

7.11.3 Recipe Tab

Recipe Values names can be inherited from the Enterprise, Site or Area level settings, or defined at the Location level.

At this level, it is possible to add configuration information regarding the tag, variance thresholds and the scripts used to evaluate them.

For more information please refer to the Recipe Production Model Configuration section.



Recipe Tab

7.11.4 Advanced Tab

At the Location level, events generated by the SPC Module are exposed and custom scripting regarding what happens when a sample is updated, scheduled or evaluated, and what should happen when an **Out Of Signal** event occurs can be defined here.

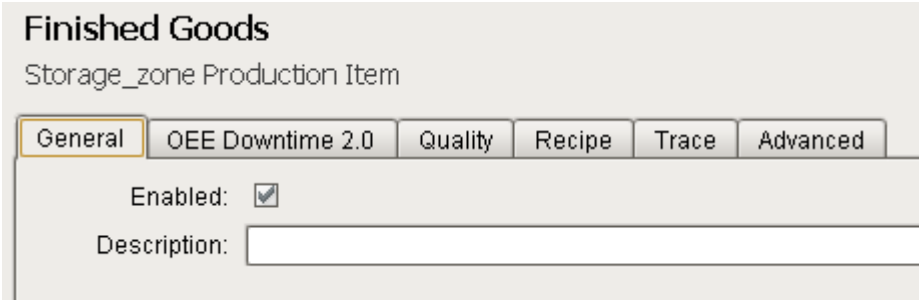
Advanced Tab

7.12 Storage Zone Settings

7.12.1 General Tab

These settings are accessed by selecting the enterprise item contained in the Production folder in the project browser and then selecting the **General** tab as shown.

By default, the Storage Zone production item is enabled. It can be disabled by un-checking the **Enabled** setting and saving the project. This will stop the MES modules from executing the Storage Zone and all other production items that are underneath it.



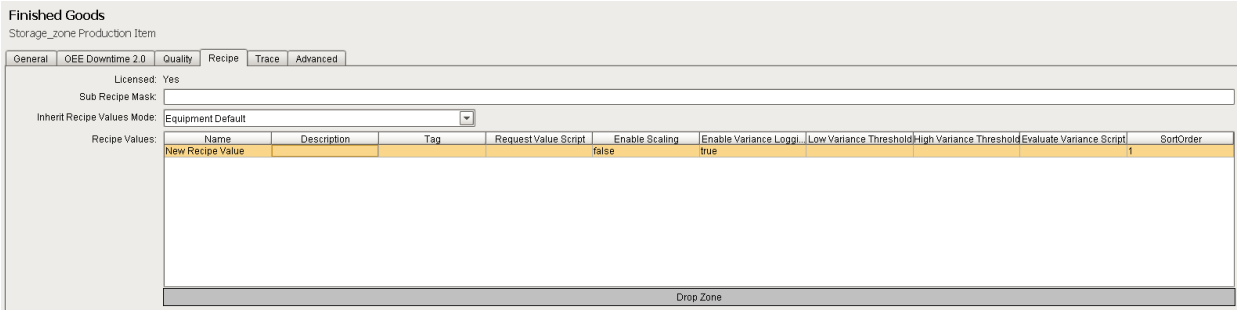
7.12.2

7.12.3 Recipe Tab

Recipe Values names can be inherited from the Enterprise, Site or Area level settings, or defined at the Storage Zone level.

At this level, it is possible to add configuration information regarding the tag, variance thresholds and the scripts used to evaluate them.

For more information please refer to the Recipe Production Model Configuration section.



7.12.4 Advanced Tab

At the Storage Zone level, events generated by the Recipe Module are exposed and custom scripting regarding what happens when a recipe is selected or canceled can be defined.

7.13 Storage Unit Settings

7.13.1 General Tab

These settings are accessed by selecting the Storage Unit item contained in the Production folder in the project browser and then selecting the **General** tab as shown.

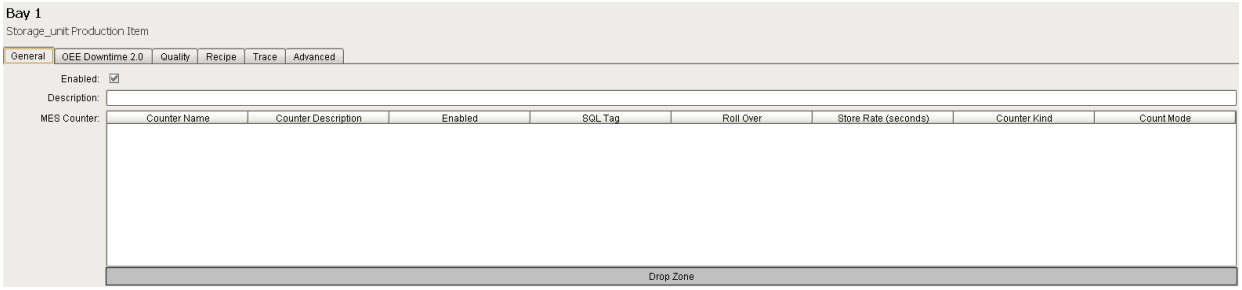
By default, the Storage Unit production item is enabled. It can be disabled by un-checking the **Enabled** setting and saving the project. This will stop the MES modules from executing the Storage Unit.

MES Counters

The MES Counters are used by the Track & Trace and OEE 2.0 Modules to associate Process Segments (Operations) with production counts. MES Counters record production counts 7/24, independent of scheduled production runs.

Counter names and the associated tag are defined in the Production Model. In the MES Management screen, the Quantity Source of Infeed and Material Process Segments can be set to use these MES counters.

More information can be found in the MES Counters page.



The quantity from the MES counters can be obtained through scripting, see system.mes.getCountValue.

The tag path for MES counters can be parameterized with "{Equipment Path}" to utilize indirection and more rapidly implement the production model.

Before: `[default]\Enterprise\Site\Area\Line\InfeedCount`

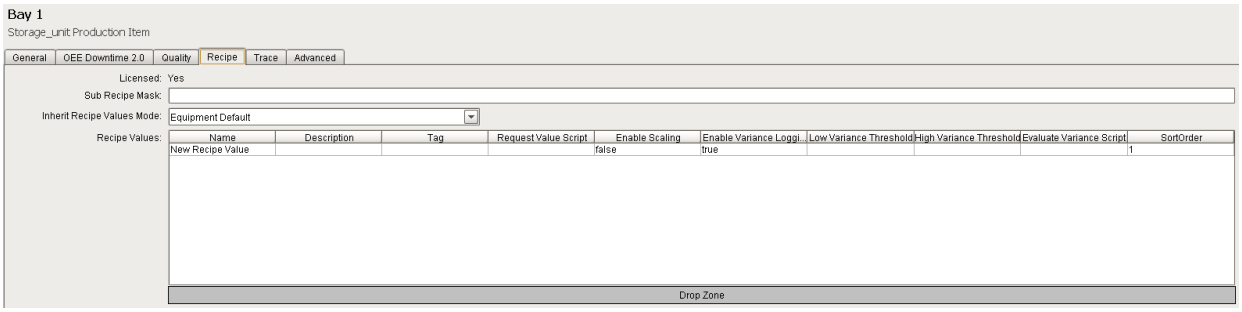
After: `{Equipment Path}\InfeedCount`

7.13.2 Recipe Tab

Recipe Values names can be inherited from the Enterprise, Site, Area or Storage Zone level settings, or defined at the Storage Unit level.

At this level, it is possible to add configuration information regarding the tag, variance thresholds and the scripts used to evaluate them.

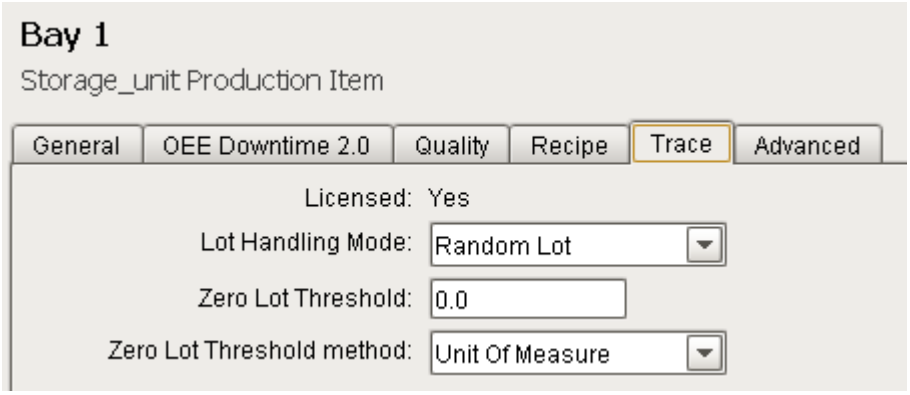
For more information please refer to the Recipe Production Model Configuration section.



7.13.3 Trace Tab

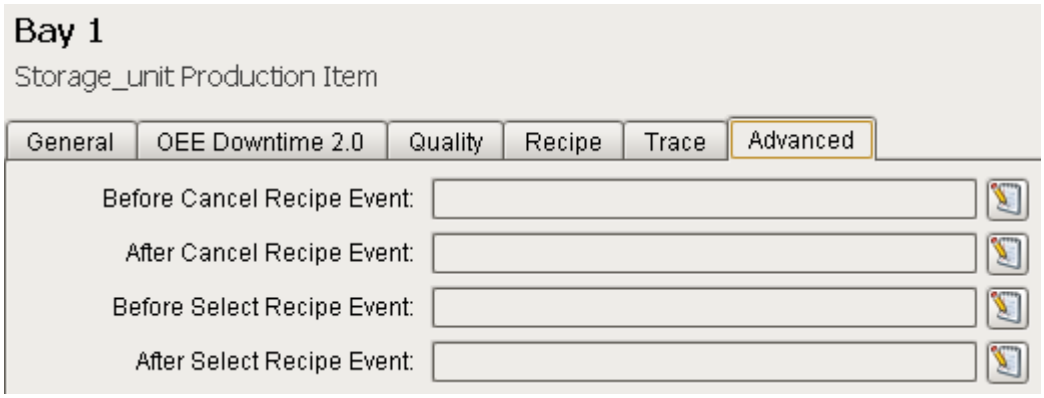
The Trace tab allows you to define the **Lot Handling Mode** for the selected line.

Refer to the Lot Handling Mode section in the Track & Trace Module help.



7.13.4 Advanced Tab

At the Storage Unit level, events generated by the Recipe Module are exposed and custom scripting regarding what happens when a recipe is selected or canceled can be defined.



7.14 Additional Factors

The OEE Module collects and logs a number of downtime and production data values. However, what if other values outside of downtime and production values are of interest? Additional factors are the solution. Additional Factors are user defined data points that are logged along with the production and downtime information. Once they are logged, they can be shown in charts, tables and reports. Additionally, other analysis can be done by filtering and/or setting up comparisons by their values.

Additional Factors can added to the Line, Cell Group and Cell Production Items in the Production Model Designer.

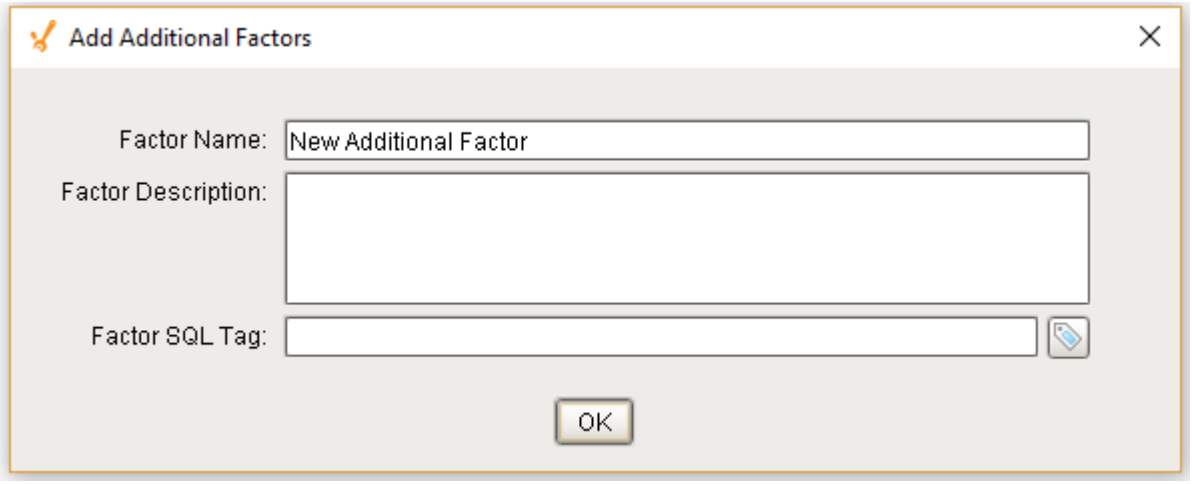
Any value that can be read from an Ignition SQLTag can be added as an additional factor. This includes values derived from scripts, or from barcode readers, databases, calculations, PLCs, etc.

Any tag can be added as an additional factor. To configure, select a Line in the production model and select the **General** tab on the right. Right click on the **Additional Factors** table and select **New**. An additional factor is simply just a name and a tag.

7.14.1 Properties

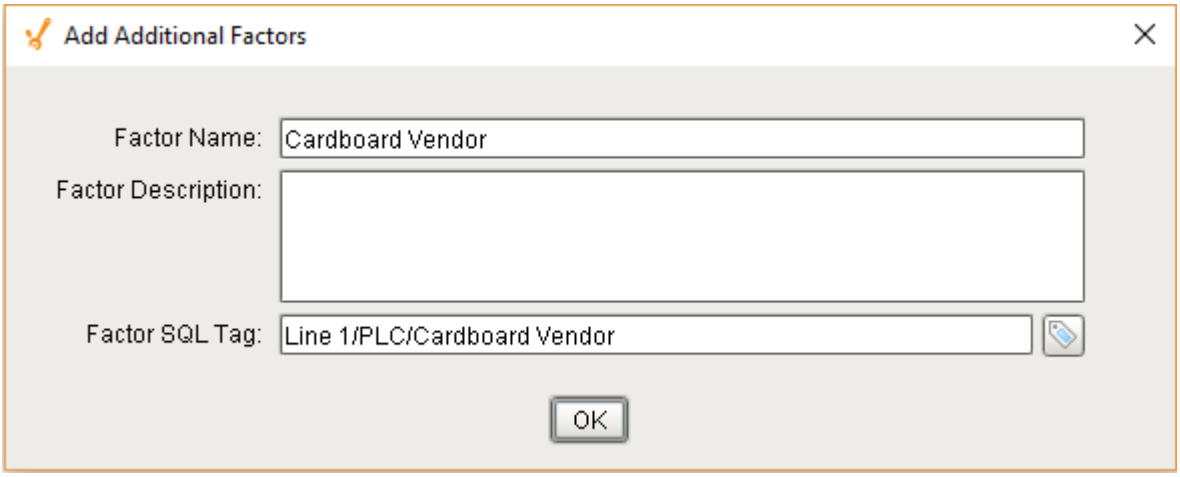
Factor Description	Optionally, this property can be set to a description for the additional factor. It is not used by the OEE Downtime and Scheduling Module other than for reference.	String
Factor Name	This reflects the name of additional factor that is configured in the designer.	String Read Only
		String

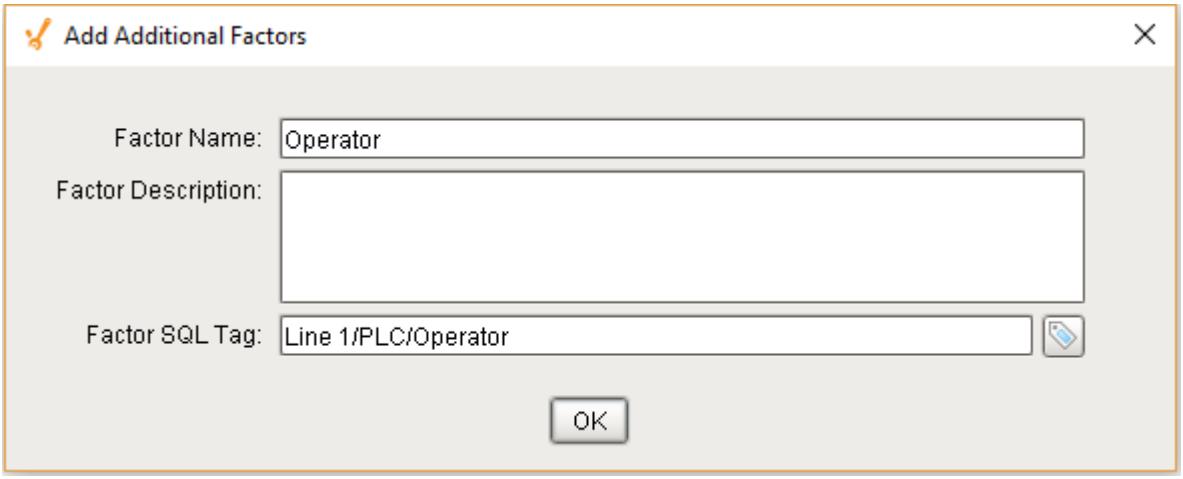
Factor SQLTag	This reflects the Factor SQLTag setting that additional factor is configured for in the designer. It is the name of SQLTag to read the factor value from.	Read Only



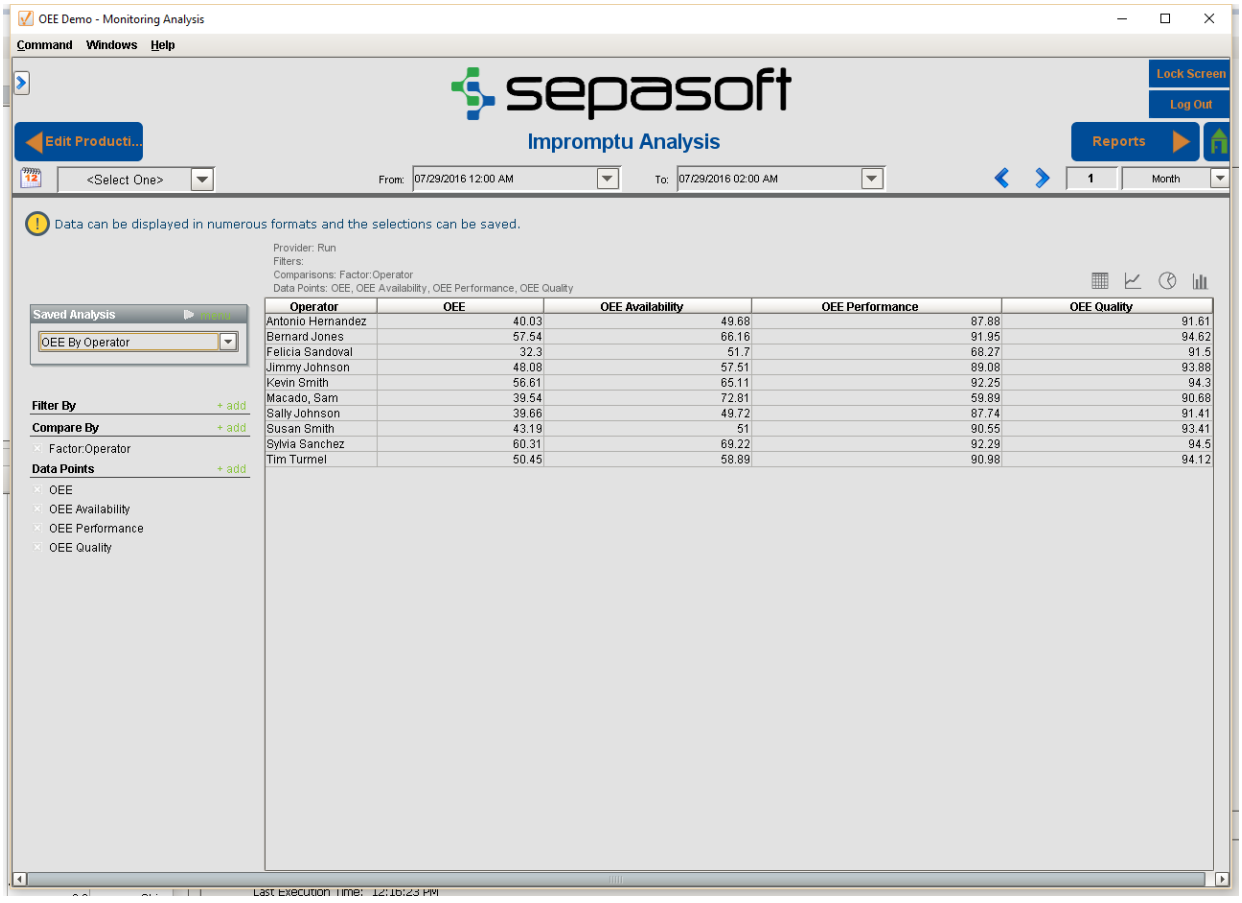
Example

In the example, we have two factors, **Cardboard Vendor** and **Operator**. The operator can select the vendor that provided the cardboard or it can be obtained from some other source. Now, OEE and downtime results can be shown for each cardboard manufacturer. This can identify quality problems with raw material that directly affect production efficiency. With the operator setup as an additional factor, the operator's name will be logged along with the production and downtime data. By doing so, OEE and downtime information can be filtered and grouped by the operator name. But this could just as well be the production crew, supervisor, maintenance crew or any other user defined value that can be monitored or entered into the system.





Adding these factors to a production line will allow us to capture the value of these tags whenever they change. In the impromptu analysis, we can then compare OEE values by our additional factor: Operator



7.15 MES Counters

The MES Counters are used to associate Process Segments (Operations) with production counts. MES Counters record production counts 7/24, independent of scheduled production runs.

Counter names and the associated tag are defined in the Production Model. In the MES Management screen, the Quantity Source of Infeed and Material Process Segments can be set to use these MES counters.

MES counters are available for the Line, Cell Group, Cell or a Storage Unit production items in the Production Model Designer.

It is recommended to set up MES Counters at the cell level in order to accurately capture counts while indexing product on the line. Addition configuration must be accomplished with the cell settings in the production model.

The quantity from the MES counters can be obtained through scripting, see system.mes.getCountValue.

7.15.1 Adding MES Counter

To configure an MES Counter, select a Line, Cell Group, Cell or a Storage Unit in the production model and select the **General** tab on the right. Right click on the **MES Counter** table and select **New**. Properties can be set through the **Add MES Counter** Window.

7.15.2 Counter Name

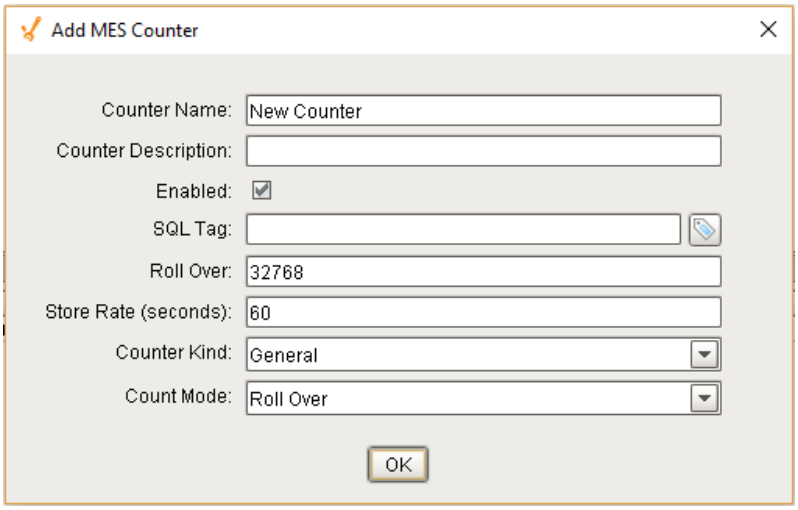
Name of the counter.

7.15.3 Counter Description

The description for the MES counter. This setting is not mandatory.

7.15.4 Enabled

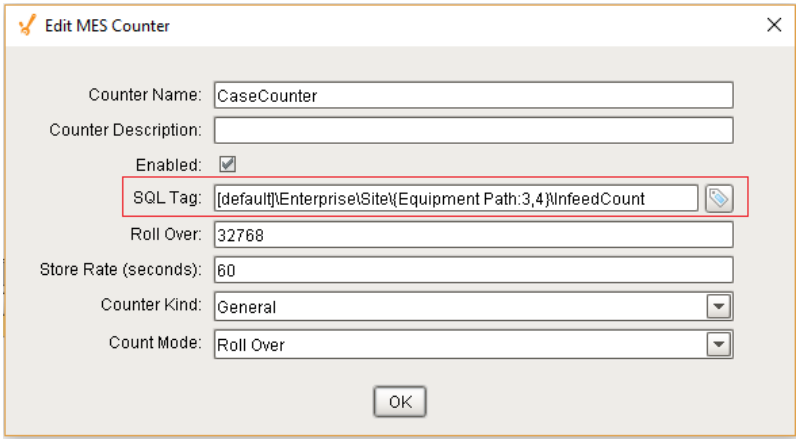
The counter can be enabled or disabled here.



7.15.5 SQL Tag

The path to the Tag Provider and ignition tag where the count value will come from is assigned to the MES counter here.

Parameterized Tag Paths can be used here which allows for indirection and exported MES Counters to be easily deployed to other equipment.



7.15.6 Roll Over

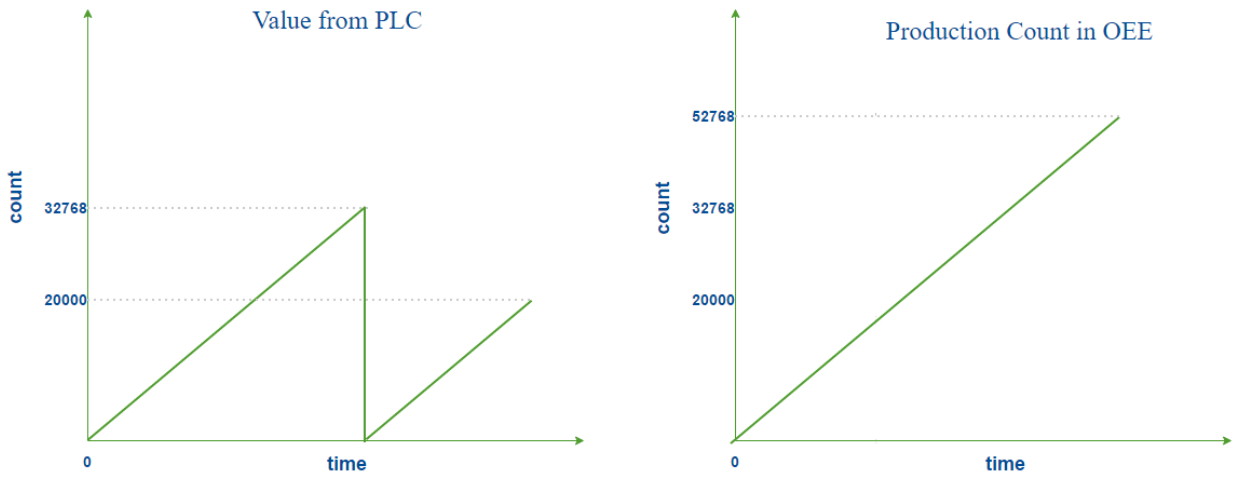
For PLC count tags that that do not get reset, they will eventually reach a finite maximum value at which point, the value will 'rollover' back to zero. The Roll Over setting allows you to define the value that should be added to the count tag whenever a roll over occurs. By default it is 32768 which equates to a 16 bit signed data value. Your setting will be dependent upon the datatype of your plc count tag.

During a production run, the incoming count value is added to the Roll Over setting multiplied by the number of times a rollover has occurred.

Example: production count value = incoming count value + 32768 * 3

Production Count	PLC Count tag	Rollover	Calculation
32700	32700	0	32700 + 32768 * 0
32750	32750	0	32700 + 32768 * 0
32918	150	1	150 + 32768 * 1
33068	300	1	300 + 32768 * 1

⚠ This value is only used when the **Count Mode** is set to **Rollover**.



Roll Over Count Mode

7.15.7 Store Rate

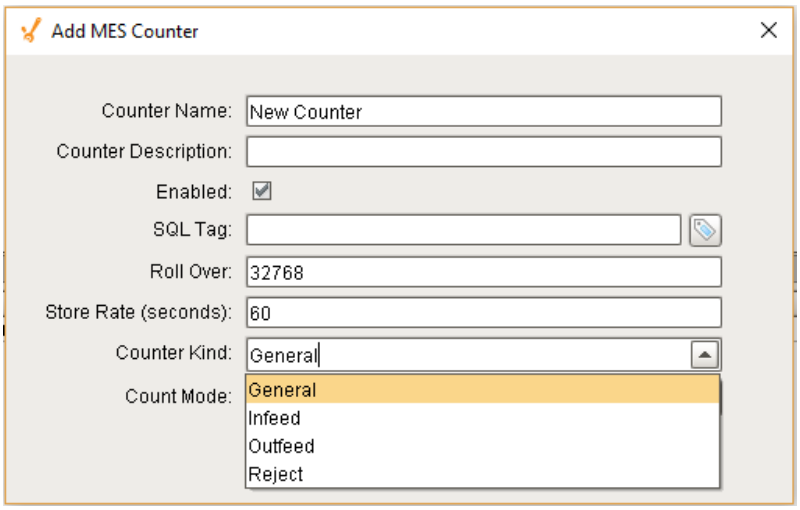
The MES counter will be captured and stored in the database after this specific interval in seconds if the value has changed If **Store Rate** is set to zero, every value change will be recorded.

7.15.8 Counter Kind

MES Counters can be set to four different kinds:

- Infeed
- Outfeed
- Reject
- General

Infeed, **Outfeed** and **Reject** kinds are used solely by the OEE module to determine which MES counter to use for OEE Performance, OEE Quality and production count information. The **General** kind can be used for any other count value.



7.15.9 Count Mode

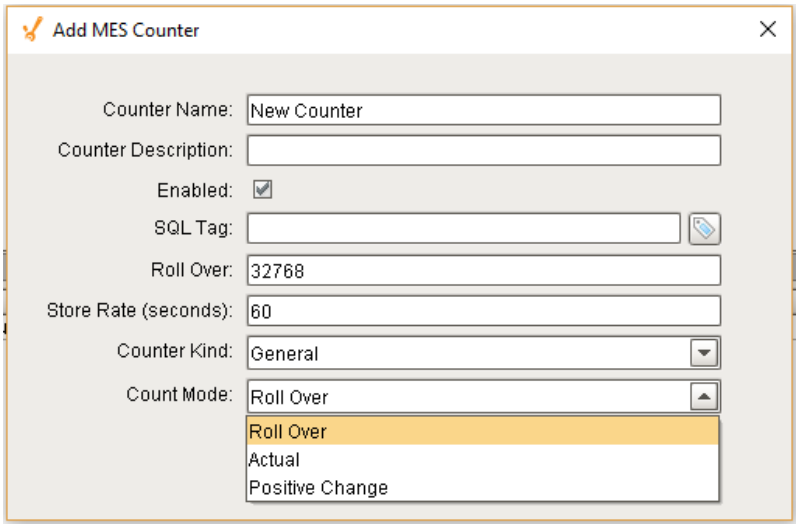
The Count Mode can be set to **Roll Over**, **Actual** and **Positive Change**.

Roll Over

See Rollover section for this count mode.

Actual

The Actual count mode simply uses whatever value is passed through the sql tag to represent the actual production counts. Production counts can go down as well as up.

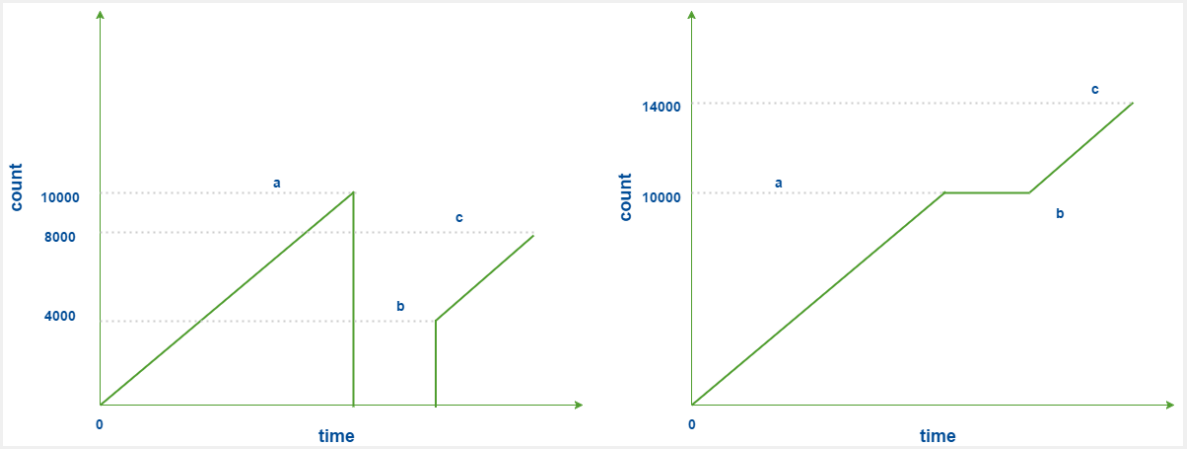


Positive Change

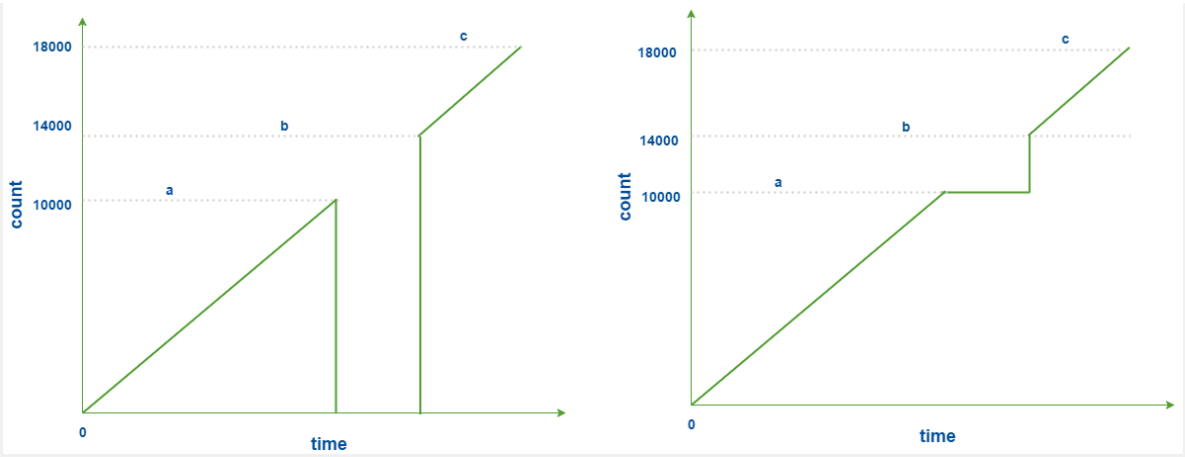
The Positive Change mode ignores any sql tag count values that are zero and will accumulate the counts. Three different cases are illustrated using the graphs shown.

PLC Count vs. OEE Count

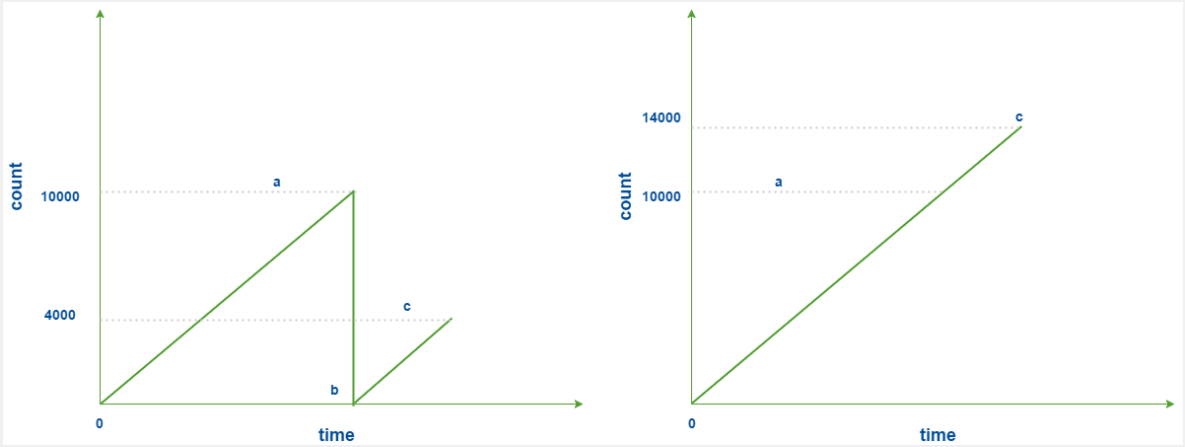
Case 1



Case 2



Case 3



Counter Rapid Development Features

Not only do counters allow for parameterization (as shown above), they also support copy, paste, import, and export features for rapid development. Configure the infeed counter for one Production Model Node (i.e. a Cell or Line) with parameterization, then copy and paste it to the other cells. Alternatively, copy and paste the Node itself and rename it for a similar effect.

7.16 Downtime Detection Mode

The OEE 2.0 Module provides a variety of methods that can be used to automatically detect and determine the underlying reason causing a production line to be down. The options have been added to accommodate the wide variety of manufacturing processes. A detailed description of each method along with the situations where it can be used is provided.

7.16.1 Equipment State

The Equipment State Downtime Detection Method provides a method for determining the Line or Sub-Line state from the provided State Tag path. Use this method if Line State will be provided from a single state tag and not derived from the state of cells on the line.

7.16.2 Initial Reason

The Initial Reason downtime detection method uses the first cell in the line or Sub-Line (cell group) that goes down for an unplanned reason as the cause for the line not being able to produce product. When a cell first goes down, the date and time is recorded. If multiple cells are down, each will have its own date and time it went down. The date and time for each down cell is looked at to determine the initial cell that went down and that cell will be assigned as the cell causing the line downtime along with its reason. If the initial cell restarts, then the other down cells are looked at in the chronological order that they went down. If there are two or more cells that went down at the same time, then the order that they appear in the designer will be used to determine the cell to blame.

If there are no cells down for an unplanned or planned reason, then the line will return to running state.

This method should be used if all cells interact with one another. If any cell is down, then all other cells have to stop. A continuous liquid mixing process where at each cell, new ingredients are added or mixed or some other action is being performed fits into this category. If one cell stops, then all other upstream cells have to stop because there is no where to put the liquid and all downstream cells have to stop because there is no liquid to process. In this case the first cell that stopped for an unplanned reason is the cause for all other cells to stop.

If a state SQL tag is defined at the line or sub-line level, any active event at the line level will be used in place of the Initial Reason.

7.16.3 Key Reason

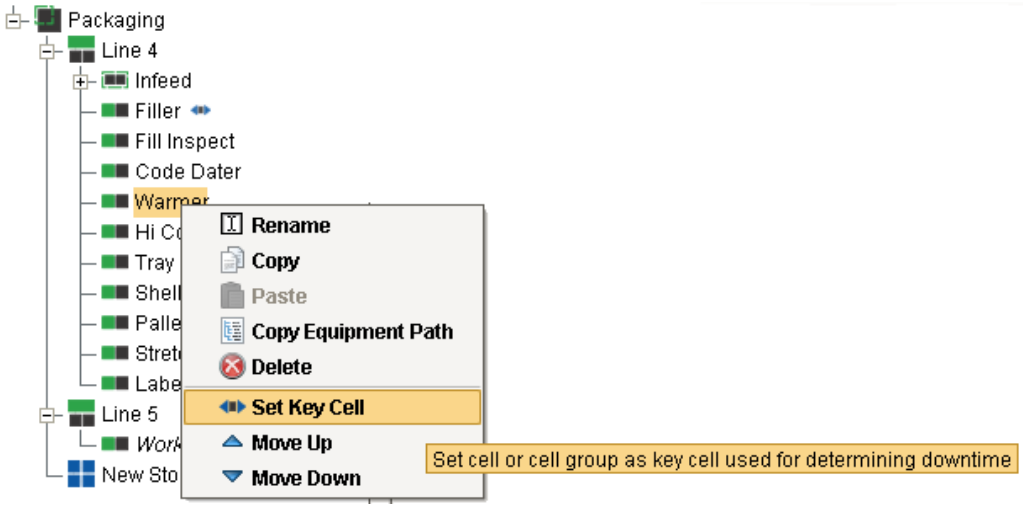
The Key Reason downtime detection method uses the flow of the line to determine the cause for the line not being able to produce product. The accurate recording of the loss production is done through this algorithm.

With Key Reason Detection method enabled for a production line, a cell under the Line or Cell Group must be defined as the **Primary Cell** . This is done by right-clicking on the cell and selecting 'Set as Key Cell'. If the primary cell is running, then the line is determined to be running. If the primary cell is down, it is assumed that this will cause the line to stop producing product. If the primary cell is down for a recordable reason (configured as Record Downtime = True), the primary cell will be considered as the cause of line or sub-line downtime. If it is down for a non-recordable downtime reason (configured as Record Downtime = False in the Equipment Manager), then cells adjacent to it will be checked for a recordable downtime state. The direction in which the cells are checked is dependent upon whether the state type of the primary cell state is set to BLOCKED or STARVED. If the state type of the primary cell state is IDLE, the primary cell state will be used for the Line state.

When the cell that caused the line downtime restarts but the primary cell has not started yet because its discharge is still blocked or starved, then the original cell and reason will still be the cause of downtime until the primary cell restarts. The concept behind this is that a faster downstream cell can go down, restart and catch up without ever causing loss of production on the line.

This method should be used for packaging lines. If the first cell on the line keeps accepting raw material, then the line will be producing product. However, in some situations, it could be the slowest machine because it cannot catch up for lost production.

The mode of the line must have **Include in OEE** set to True for Key reason detection method to work.



Cell Priority

When the primary cell is either blocked or starved, the cell closest to the primary cell that went down for a recordable downtime reason is latched and used as the cause of line downtime, even if that cell is back up and running at the time the primary cell goes down. If the primary cell is starved, the OEE module will look upstream from the primary cell for the source of the problem. Likewise, if the primary cell is blocked, the module will look downstream for the cause. If the primary cell goes down for a recordable downtime reason, then the primary cell downtime reason becomes the downtime reason.

Neighbor Priority

Latches onto the most downstream cell that went down for a recordable downtime reason.

7.16.4 Parallel Cells

This Downtime Detection method is used on Cell Groups and uses the Minimum Cells Running Threshold to determine how many cells in the Cell Group must be running in order for the Cell Group to be considered as Running.

7.17 OEE Downtime 2.0 Tab

The OEE Downtime 2.0 tab is specific to the OEE module and is available for the Line, Cell Group and Cell Production Items. A number of configuration settings are provided that can be used to obtain equipment mode, state and count values from ignition tags (whether plc tags, memory or expression tags).

7.17.1 Downtime Detection Mode

How line downtime is determined can be changed based on the selected Downtime Detection Mode. Valid options for the downtime detection mode are...

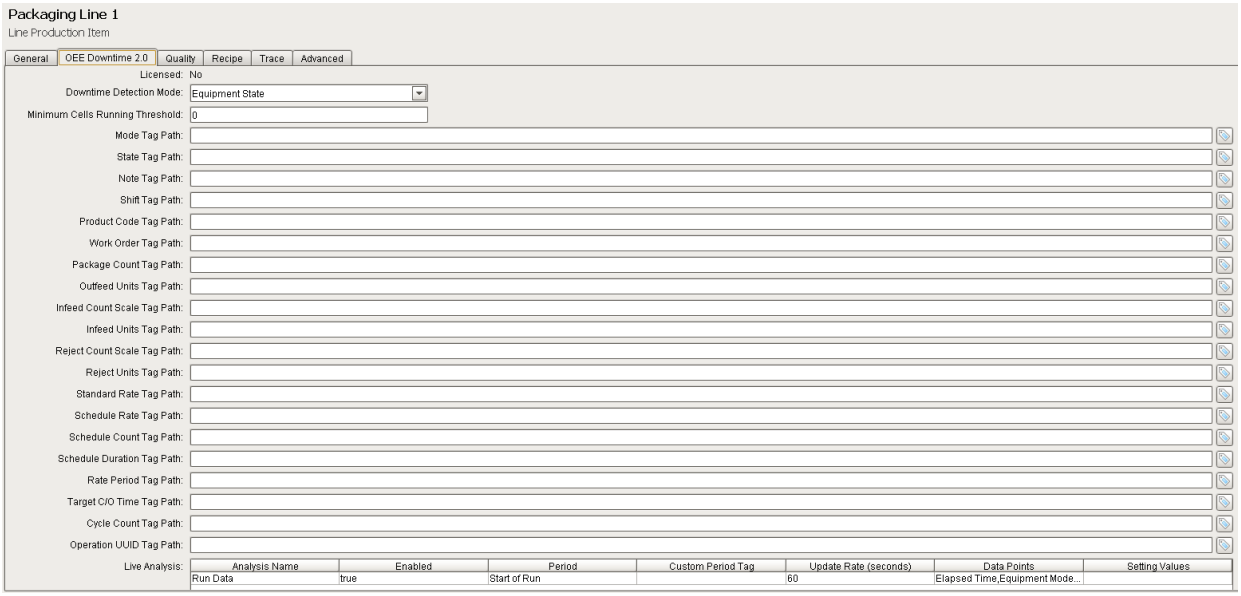
- Equipment State
- Key Reason (Cell Priority)
- Key reason (Neighbor Priority)
- Initial Cell
- Parallel Cells

Refer to Downtime Detection Mode for more information on the various Downtime Detection Methods.

> ⚠ Downtime Detection Mode is only available for the Line and Cell Group Production Item.

7.17.2 Minimum Cells Running Threshold

Minimum Cells Running Threshold determines how many cells in the Line (or Cell Group) must be running in order for the Line (or Cell Group) to be considered as Running.



7.17.3 Tag Collector Paths

Tag Collectors are provided to allow any of the parameters needed to drive OEE Metrics to be provided externally to the OEE module. Virtually all the Tag Collectors can be left blank in which case the OEE engine will determine the value from product code configuration information as defined in the OEE Material Manager or from internal calculations. Exceptions to this would be the equipment state and counts where needed.

> When adding a tag to a Tag Collector, you must use memory tags. The Production Model will write values to any tags defined here as well as read the value whenever it changes from an external source.

Tag Collector Path	Tag Type	Data Type	Description
Mode	Memory	Integer	The Mode Tag, if provided, will be written to by the Production Model whenever the equipment mode changes based on Material Production Settings. The value of the mode tag can also be written to whenever the Mode value changes either indirectly from a plc tag (via tag change event) or from the HMI. The value of the Mode tag will be recorded for the current mode.
State	Memory	Integer	The State Tag path will generally come from a plc as the source of the current equipment state. Exceptions to this are at the Line level when using a downtime detection method other than Equipment State.
Downtime Note	Memory	String	Apart from scripting and Downtime table component, the downtime notes can be added by using tags.
Shift	Memory	String	When left blank, shifts defined in the Ignition Schedule Management component and defined in the Equipment Manager for a line will be used to determine the current shift. If a tag is provided here, whatever value is in the tag e. g. 'Shift A' will be recorded for the current shift.
	Memory	String	

Tag Collector Path	Tag Type	Data Type	Description
Product Code			When left blank, the Product Code currently running on the line will be determined from the scheduled run as selected by the Scheduler or Run Director component. When a tag path is provided, the product code for the line or equipment (cell) will be determined from the value of the tag.
Work Order	Memory	String	When left blank, the Work Order currently running on the line will be determined from the scheduled run as selected by the Scheduler or Run Director component. When a tag path is provided, the product code for the line or equipment (cell) will be determined from the value of the tag.
Package Count	Memory	Float	When left blank, the Package Count will be determined from the Package count setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Package Count for the line or equipment (cell) will be determined from the value of the tag. For more information on the Package Count, refer to the Material Production Settings section.
Line Outfeed Units	Memory	String	When left blank, the Line Outfeed Units will be determined from the Line Outfeed Units setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Line Outfeed Units for the line or equipment (cell) will be determined from the value of the tag. For more information on the Line Outfeed Units, refer to the Material Production Settings section.
Infeed Count Scale	Memory	Float	When left blank, the Infeed Count Scale will be determined from the Infeed Count Scale setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Infeed Count Scale for the line or equipment (cell) will be determined from the value of the tag. For more information on the Infeed Count Scale, refer to the Material Production Settings section.

Tag Collector Path	Tag Type	Data Type	Description
Line Infeed Units	Memory	String	When left blank, the Line Infeed Count Scale will be determined from the Infeed Count Scale setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Infeed Count Scale for the line or equipment (cell) will be determined from the value of the tag. For more information on the Line Infeed Units, refer to the Material Production Settings section.
Reject Count Scale	Memory	Float	When left blank, the Reject Count Scale will be determined from the Reject Count Scale setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Reject Count Scale for the line or equipment (cell) will be determined from the value of the tag. For more information on the Reject Count Scale, refer to the Material Production Settings section.
Line Reject Units	Memory	String	When left blank, the Line Reject Units will be determined from the Line Reject Units setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Line Reject Units for the line or equipment (cell) will be determined from the value of the tag. For more information on the Line Reject Units, refer to the Material Production Settings section.
Standard Rate	Memory	Float	When left blank, the Standard Rate will be determined from the Standard Rate setting for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Standard Rate for the line or equipment (cell) will be determined from the value of the tag. For more information on the Standard Rate, refer to the Material Production Settings section.
Schedule Rate	Memory	Float	When left blank, the Schedule Rate will be determined from the Schedule Rate setting for the currently running Product Code (Material) as defined in the Material Manager. When a

Tag Collector Path	Tag Type	Data Type	Description
			tag path is provided, the Schedule Rate for the line will be determined from the value of the tag. For more information on the Schedule Rate, refer to the Material Production Settings section. ⚠ Schedule Rate Tag Path is only available for the Line Production Item.
Schedule Count	Memory	Integer	When left blank, the Schedule Count will be determined from the scheduled run as selected by the Scheduler or Run Director component. When a tag path is provided, the Schedule Count for the line or equipment (cell) will be determined from the value of the tag. The Schedule Count provides the number of units scheduled to be produced. ⚠ Schedule Count Tag Path is only available for the Line Production Item.
Schedule Duration	Memory	Integer	When left blank, the Schedule Duration will be determined from the scheduled run as selected by the Scheduler or Run Director component. When a tag path is provided, the Schedule Duration for the line or equipment (cell) will be determined from the value of the tag. The Schedule Duration provides the expected runtime required for the number of units scheduled to be produced and is calculated by the Schedule Rate.
Rate Period	Memory	String	When left blank, the Rate Period will be determined from the Rate Period setting for the currently running Product Code (Material) as defined in the Material Manager. When a

Tag Collector Path	Tag Type	Data Type	Description
			tag path is provided, the Rate Period for the line or equipment (cell) will be determined from the value of the tag. For more information on the Rate Period, refer to the Material Production Settings section.
Target C /O Time	Memory	Integer	When left blank, the Target C/O (Changeover) Time will be determined from the Changeover settings for the currently running Product Code (Material) as defined in the Material Manager. When a tag path is provided, the Target C/O (Changeover) Time for the line or equipment (cell) will be determined from the value of the tag. For more information on the Target C/O (Changeover) Time, refer to the Material Production Settings section. ⚠ Target C/O Time Tag Path is only available for the Line Production Item.
Cycle Count	Memory	Float	When left blank, the Cycle Count will be determined by the OEE Module. When a tag path is provided, the Cycle Count for the line or equipment (cell) will be determined from the value of the tag.
Operation UUID	Memory	String	When left blank, the Operation UUID will be determined from the currently running Operation on the Line or equipment (cell). When a tag path is provided, the Operation UUID for the line or equipment (cell) can be determined from the value of the tag. The purpose for this tag is to be able to provide OEE analysis data when production runs are not scheduled or started using the Run Director or Schedule Selector components, or scripting functions. In this case a tag can be used to provide a Run Identifier value i.e. Run_4253_XX. The Analysis Selector provides the ability to pull the Operation UUID as part of analysis, whether it is an internally generated Operation UUID or a passed Run Identifier.

Tag Collector Paths can be parameterized with {Equipment Path} to utilize indirection and more rapidly implement the production model. See Parameterized Tag Paths for more details.

7.17.4 Live Analysis

Live Analysis provides a flexible way of customizing your application to provide a set of real-time tag values that can be accessed from the Ignition designer and used in your application to provide real-time production monitoring. Live Analysis is configured in the OEE 2.0 Downtime tab of the Production Model Designer for the Line, Cell Group and Cell production items. When a Live Analysis is created, a corresponding set of tags is created in the MES Tag Provider that provide the real-time status of those datapoints based upon the Period defined for the Live Analysis. You can create multiple Live Analysis and use those tags to drive HMI displays.

To create a new Live Analysis:

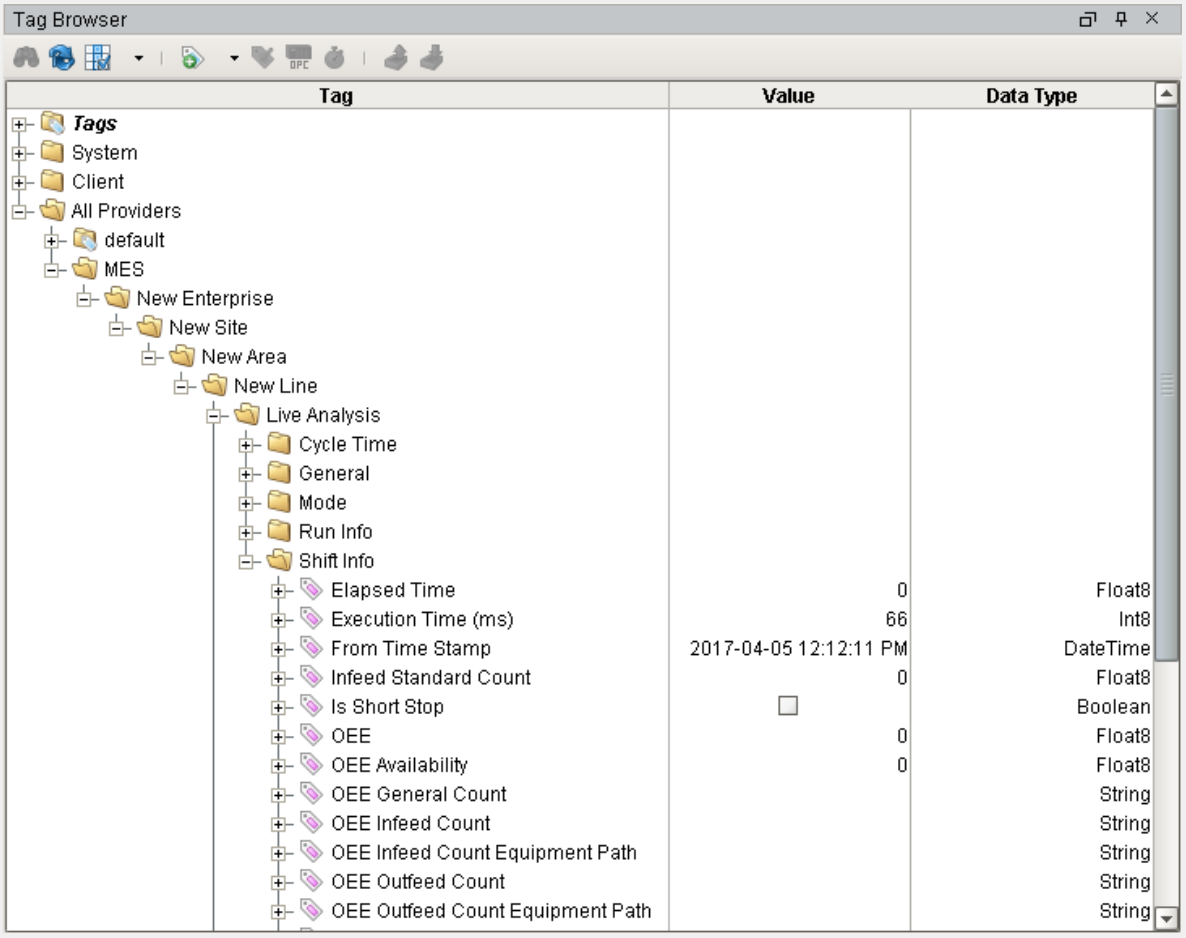
- Right click on the Live Analysis panel on the OEE 2.0 Downtime Tab in the Production Model Designer.
- Provide a Name
- Select the Period that the Live Analysis datapoints will return a value for. Valid options are Shift, Day (Midnight), Day (Production), Start of Run, Top of Hour, Custom Period Tag
- Select the frequency for how often the tag values will be updated. Default value is 60 seconds. Minimum value is 10 seconds
- Select the desired Data Points
- Add any further Settings Values required

You cannot select all Data Points in one Live Analysis. The maximum length string for Data Points is 1024 characters

Live Analysis:

Analysis Name	Enabled	Period	Custom Period Tag	Update Rate (seconds)	Data Points	Setting Values
Cycle Time	true	Shift		60	Average Normal Cycl...	
General	true	Shift		60	Delta Time Stamp,Eq...	
Mode	true	Shift		60	Equipment Mode Na...	
Run Info	true	Start of Run		60	Equipment Cell Orde...	
Shift Info	true	Shift		60	Elapsed Time,Infeed ...	

Live Analysis Settings Panel in the OEE 2.0 Downtime Tab



MES Tag Provider Live Analysis Tags

Live Analysis Settings

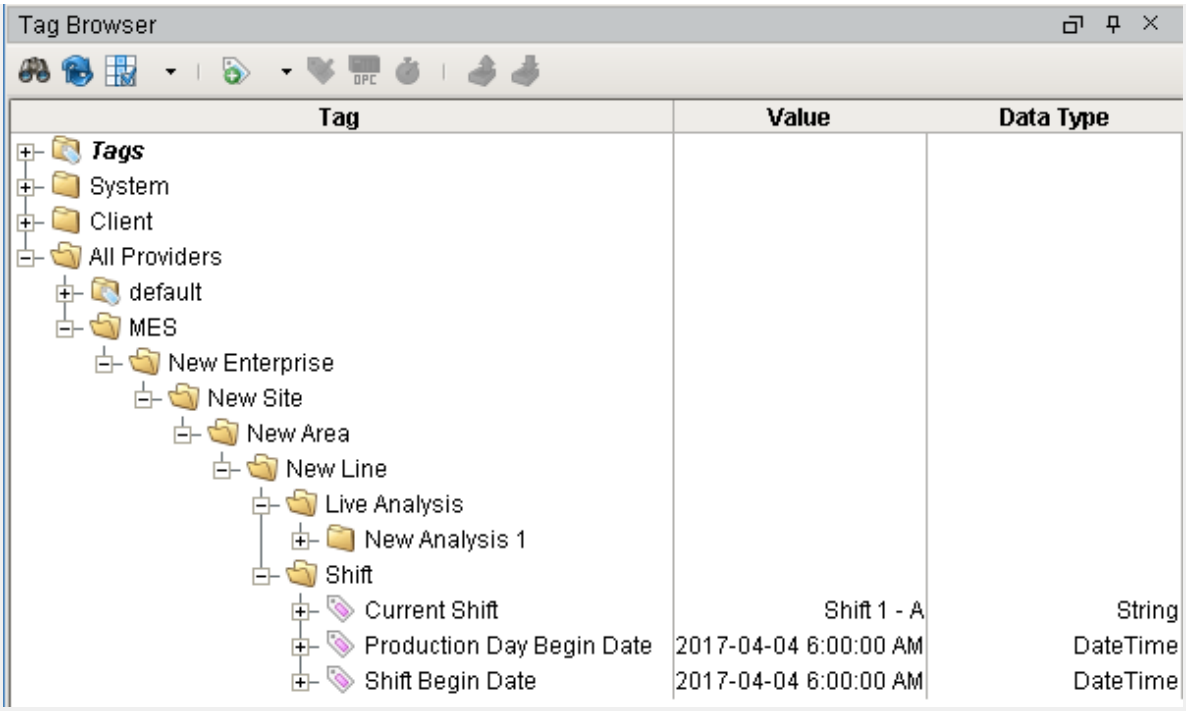
Setting	Description
Analysis Name	The name for the live analysis.
Enabled	The live analysis can be enabled or disabled with this setting.

Setting	Description
Period	The duration of analysis can be set by **Shift**, **Day (midnight)**, **Day (production)**, **Start of Run**, **Top of Hour** or **Custom Period Tag**.
Custom Period Tag	A tag can be assigned to define the start datetime for a custom period. The end time will be the current time. It takes value in the date time data type. Example for a valid value for the custom period tag is: **`2017/04/04 14:00:00`**
Update Rate	The rate in seconds by which the live analysis is updated. The minimum update rate is 60 seconds.
Data Points	Data points allows you to pick and choose the values you wish to access through tags. See the table below for the listing of available data points.

Shift Data Points

When creating a Live Analysis, the following shift data points will be automatically created.

Data Point	Data Type	Description
Current Shift	String	The currently running shift as defined in the Ignition Schedule Management component or passed from the Shift Tag Collector path
Production Day Begin Date	DateTime	Production start time
Shift Begin Date	DateTime	Start time of the current shift



Analysis Data Points and Settings are used by Live Analysis, the MES Analysis Selector and MES Analysis Controller components, the MES Analysis Data Source for reporting and the MES Analysis Settings object.

Equipment Data Points

Data Point	Data Type	Description
Equipment		
Equipment Cell Order	Int4	Integer value that determines the cell order of the equipment within the line. Is set to *null* for the line. Is set to 0 for first cell within each cell group
Equipment Name	String	Name of the equipment as defined in the production model
Equipment Note	String	Any note that has been recorded for this piece of equipment through the Note tag collector path in the Production model will be exposed here.

Data Point	Data Type	Description
Equipment Operation Begin	DateTime	Start Date time of the currently running operation on this equipment
Equipment Operation Sequence	Int4	The ordinal number (integer) of operation
Equipment Path	String	Production model path for this equipment
Equipment Type	String	Can be Line, Cell Group or Cell
Execution Time (ms)	Int8	Time taken to execute and update the Live Analysis. Used mainly for performance debugging
From Time Stamp	DateTime	Start Date Time of current data point results
Infeed Units	String	See Infeed Units for more details
Is Key Cell	Boolean	See Key Reason for more details
Operation UUID	String	Unique Identifier for currently running operation
Outfeed Units	String	See Outfeed Units for more details
Product Code	String	Product code currently being processed on this equipment
Rate Period	String	See Rate Period for more details

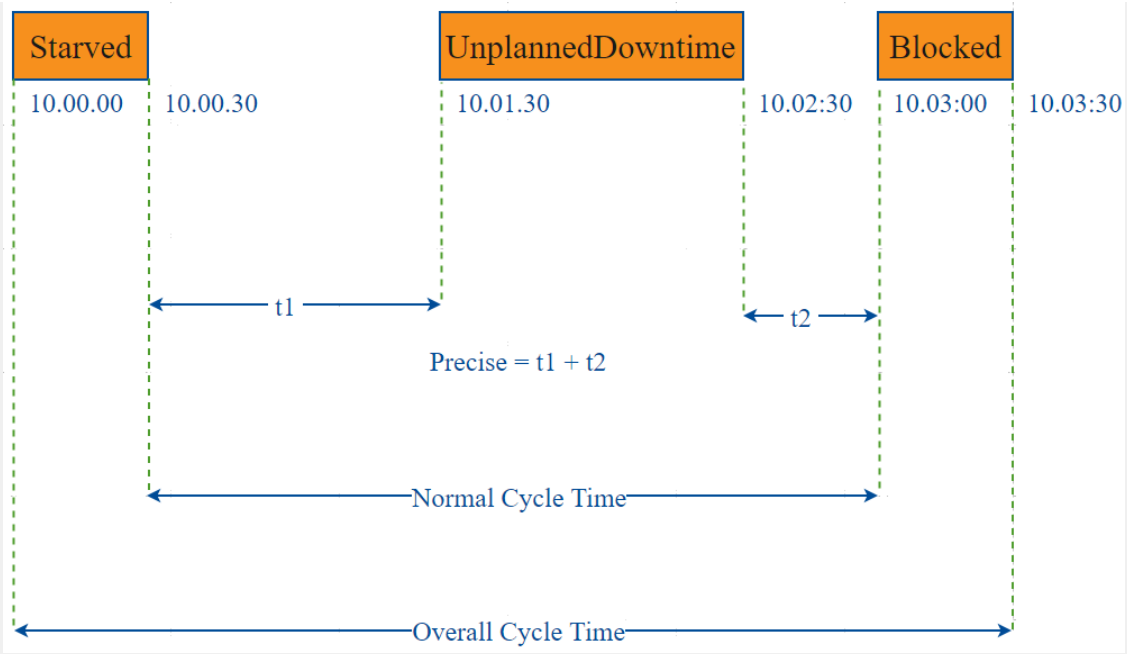
Data Point	Data Type	Description
Reject Units	String	See Reject Units for more details
To Time Stamp	DateTime	End Date Time of current data point results
Work Order	String	Work order currently being processed on this equipment

Equipment Count Data Points

Data Point	Data Type	Description
Equipment\Count		*Any defined counters for the production item will also appear in this folder
Equipment Infeed Scale	Float8	See Infeed Count Scale for more details
Equipment Package Count	Float8	See Package Count for more details
Equipment Reject Scale	Float8	See Reject Count Scale for more details
Outfeed-Material Out	String	Value of the default MES Counter used for OEE outfeed count

Equipment Cycle Time Data Points

The Cycle Time data points provides a number of metrics that can be used to measure the amount of time required to produce one piece. It is often used to gain an understanding of variations in production. Live Analysis provides Target, Normal, Overall and Precise Cycle Time metrics.



Data Point	Data Type	Description
Equipment\Cycle Time		
Relative Cycle Count	String	Relative Cycle Count is how many occurred for the compare by.
Target Cycle Time	Float8	Also known as Takt time, it is how often a piece must be produced to meet customer demand. It is often used to pace a production line, and it is a calculated number in seconds.
Total Cycle Count	String	Total Cycle Count is accumulative, it is sum total of all the cycle count.
Equipment\Cycle Time\Normal		**Normal Cycle Time is the actual cycle ignoring the equipment states like starved, blocked, etc.**
Average Normal Cycle Time	Float8	Average Normal cycle time in seconds for the time period selected
	Float8	

Data Point	Data Type	Description
Max Normal Cycle Time		Max Normal cycle time in seconds for the time period selected
Min Normal Cycle Time	Float8	Min Normal cycle time in seconds for the time period selected
Normal Cycle Time	Float8	Normal Cycle Time in seconds is the actual cycle ignoring the equipment states like starved, blocked, etc.
Equipment\Cycle Time\Overall		**Overall Cycle Time is the cycle including states like downtime, starved, blocked, etc.**
Average Overall Cycle Time	Float8	Average Overall cycle time in seconds for the time period selected
Max Overall Cycle Time	Float8	Max Overall cycle time in seconds for the time period selected
Min Overall Cycle Time	Float8	Min Overall cycle time in seconds for the time period selected
Overall Cycle Time	Float8	Overall Cycle Time in seconds is the cycle including states like downtime, starved, blocked, etc.
Equipment\Cycle Time\Precise		**Precise Cycle Time is the cycle time ignoring all the equipment states**
Average Precise Cycle Time	Float8	Average Precise cycle time in seconds for the time period selected
Max Precise Cycle Time	Float8	Max Precise cycle time in seconds for the time period selected
Min Precise Cycle Time	Float8	Min Precise cycle time in seconds for the time period selected

Data Point	Data Type	Description
Precise Cycle Time	Float8	Precise cycle time in seconds excluding states like planned downtime, unplanned downtime, starved and blocked.

Line Data Points

The Line Data points returns data for the line regardless of the Equipment the Live Analysis has been set up for.

Data Point	Data Type	Description
Line /Downtime		
Line Downtime Equipment Name	String	Name of the equipment that is responsible for causing line downtime
Line Downtime Equipment Path	String	Production model equipment path for equipment that is responsible for causing line downtime
Line Downtime Event Sequence	Int4	Every downtime event on the line is provided with an incrementing sequence number
Line Downtime Note	String	Note entered at the Line level
	Int4	Number of downtime events for the selected period.

Data Point	Data Type	Description
Line Downtime Occurrence Count		
Line Downtime Reason	String	The line or cell group (sub line) downtime reason. 1. When the line is down the Line Downtime Reason is the same as the Line State Name. 2. When the line is up the Line Downtime Reason is blank.
Line Downtime Reason Path	String	The full reason name for line or cell group (sub line) downtime reason. Line State name including State Class i.e. *Default/Cell Faulted*
Line Downtime Reason Split	Boolean	The line downtime reason split indicator. True is current downtime event has been split into multiple downtime events
Line Downtime State Time Stamp	DateTime	The time stamp for the equipment state change of the cell group (sub line) or cell that caused the line down time even.
Line /Meantime		
Line MTBF	Float8	The calculated Meantime (minutes) Between Failure for the selected period. Refer to Setting Up Equipment States - Meantime Metrics for more details.

Data Point	Data Type	Description
Line Meantime Metrics Enabled	Boolean	Returns if Meantime metrics have been enabled for this equipment.
Line /Schedule		
Line Schedule Available	Boolean	True if this operation was scheduled
Line Schedule Available Time	Float8	Time in minutes for available production time adjusted for line schedule availability and mode.
Line Standard Count	String	Amount of product that should have been produced based on the line schedule available time and line standard rate
Line Standard Count Variance	String	Variance between standard count and actual count
Line Target Count	String	Amount of product that should have been produced based on the line schedule available time and line schedule rate
Line Target Count Variance	String	Variance between line scheduled count and line OEE outfeed count.
Schedule Rate	Float8	See Schedule Rate for more details
Line/State		

Line State Duration	Float8	The line or cell group (sub line) downtime event duration in minutes.
Line State Event Begin	DateTime	The line or cell group (sub line) downtime event begin date time.
Line State Event End	DateTime	The line or cell group (sub line) downtime event end date time.
Line State Event Sequence	Int4	The equipment state event sequence number.
Line State Name	String	The line or cell group (sub line) state. 1. When the line is down the Line Downtime Reason is the same as the state name. 2. When the line is up the Line Downtime Reason is blank.
Line State Override Scope	String	The state override scope for a line or cell group (sub line). See Setting Up Equipment - Override Scope for more details
Line State Override Type	String	The state override type for a line or cell group (sub line). See Setting Up Equipment - Override for more details
Line State Type	String	The line or cell group (sub line) state type. See Setting Up Equipment - State Type for more details
Line State Value	Int4	

Data Point	Data Type	Description
		The line or cell group (sub line) downtime state code. See Setting Up Equipment - State Code for more details

Equipment Mode & State Data Points

Data Point	Data Type	Description
Equipment /Mode		
Equipment Mode Name	String	Name of the current mode. See Setting Up Equipment Modes for more details
Equipment Mode Type	String	Name of the current mode type. See Setting Up Equipment Modes for more details
Equipment Mode Value	Int4	Name of the current mode. See Setting Up Equipment Modes for more details
Mode Begin Time	DateTime	Start time of the current mode
Mode Duration	Float8	Duration of the current mode in minutes
Mode End Time	DateTime	End time of the current mode
OEE Enabled	Boolean	See Setting Up Equipment Modes - OEE Enabled for more details
Production Counts Enabled	Boolean	See Setting Up Equipment Modes - OEE Enabled for more details

Data Point	Data Type	Description
Equipment /State		
Equipment Original State Value	Int4	The original value of equipment state tag collector before it is updated by using MES Value Editor component or scripting
Equipment State Name	String	Current state name
Equipment State Path	String	Production model equipment path for equipment that is responsible for causing line downtime
Equipment State Split	Boolean	True is current downtime event has been split into multiple downtime events
Equipment State Type	String	See Setting Up Equipment - State Type for more details
State Begin Time	DateTime	Start time of the current state
State Duration	Float8	Duration of current state in minutes
State End Time	DateTime	End time of the current state

Equipment Meantime Data Points

Data Point	Data Types	Description
Equipment/Meantime		
Equipment MTBF	Float8	

Data Point	Data Types	Description
		The Mean Time (minutes) Between Failure for the selected period
Equipment Meantime Metrics Enabled	Boolean	True if Equipment Meantime Metrics are enabled for the current equipment state

Equipment General Data Points

Data Point	Data Types	Description
Equipment /General		
Delta Time Stamp	Float8	Time gap (minutes) between the rows of data.
From Time Stamp	DateTime	Start time (minutes) of the current period
Shift	String	Name of the current shift as set by the Ignition Schedule Management component and defined for the current line or by the value passed in the equipment shift tag collector
Shift Day Text	String	Name of the current day
Shift Day of Month	Int4	Int value of the current month
Shift Day of Week	Int4	Int value of the current day of the week
Shift Day of Year	Int4	Int value of the current day of the year

Data Point	Data Types	Description
Shift ISO Week of Year	Int4	Int value of the ISO week of the year
Shift Month Text	String	Name of the current month
Shift Month of Year	Int4	Int value of the current month of the year
Shift Start Date	DateTime	Start time of the current shift
Shift Week of Month	Int4	Int value of the current week of the month
Shift Week of Year	Int4	Int value of the current week of the year
Shift Year	Int4	Int value of the current year
To Time Stamp	DateTime	Endtime (minutes) of the current period

Equipment OEE Data Points

Data Point	Data Type	Description
Equipment/OEE		
Elapsed Time	Float8	Elapsed Time of current operation
OEE	Float8	OEE value for selected period

Data Point	Data Type	Description
OEE General Count	Long	Any count value other than infeed, outfeed, reject and waste value for the selected time period
OEE Infeed Count	Long	Equipment infeed count value for the selected period
OEE Infeed Count Equipment Path	String	Infeed count tag collector path
OEE Outfeed Count	Long	Equipment outfeed count value for the selected period
OEE Outfeed Count Equipment Path	String	Outfeed count tag collector path
OEE Reject Count	Long	Equipment reject count value for the selected period
Planned Downtime	Float8	Planned Downtime duration (Double) for selected period
Runtime	Float8	Planned Downtime duration (Double) for selected period
Short Stop Time	Float8	Short stop duration (Double) for selected period
Standard Rate	Float8	See standard rate for more details
Target Changeover Time	Float8	Amount of time in minutes set for Target Changeover. See Changeover Duration for more details
Unplanned Downtime	Float8	Unplanned Downtime duration (Double) for selected period

Data Point	Data Type	Description
Equipment/OEE /Availability		
Is Short Stop	Boolean	True if current equipment state is consider a shortstop. See Short Stop Threshold for more details
OEE Availability	Float8	OEE Availability value for selected period
Equipment/OEE /Performance		
OEE Performance	Float8	OEE Performance value for selected period
Infeed Standard Count	Float8	Calculated expected infeed based on standard rate
Equipment/OEE /Quality		
OEE Quality	Float8	OEE Quality value for selected period

Setting Values

The analysis results that are returned can be modified through the use of settings. Setting values provide a number of keywords as listed below.

Format for entering the keywords is k*eyword1=True, keyword2=100.0, keyword3=10.*

> ⚠ Settings like Enable Totalized Mode, Include Future, Last Values and Rollup Time span is meant for analysis selector and not for live analysis

Setting	Description	Use	Example
Date Format		All	

Setting	Description	Use	Example
	Date format fields can be customized with this setting e.g. 'YYYY/MM/dd hh:mm:ss a'		Date Format = `2017/04/12 19:45:30`
Enable Totalized Mode	This setting accumulates the count. Useful for charts where you wish to display the accumulated production count over time	Not valid for Live Analysis	Enable Totalized Mode = True
Include Future	Allows for count values to be calculated in the future. Useful for charts where you want to display target counts for future runs	Not valid for Live Analysis	Include Future = True
Last Values	Only the latest values are shown.	Not valid for Live Analysis	Last Values = True
OEE Availability Cap	The maximum value calculated can be capped with this setting	All	OEE Availability Cap = 100.0
OEE Performance Cap	The maximum value calculated can be capped with this setting	All	OEE Performance Cap = 100.0
OEE Quality Cap	The maximum value calculated can be capped with this setting	All	OEE Quality Cap = 100.0
Rollup Time Span	If the time (seconds) between downtime events is less than the rollup time and it is the same equipment and reason, then it will rollup the event into one row in the results and will increase the occurrence count.	Not valid for Live Analysis	Rollup Time Span = 30

Setting	Description	Use	Example
Row Limit	The analysis can be limited to a certain number of rows.	All	Row Limit = 10

7.18 Parameterized Tag Paths

The tag path for MES counters and Tag Collector Paths in the OEE 2.0 Downtime Tab can be parameterized with "{Equipment Path}" to utilize indirection and provide for a more rapid development of the production model.

7.18.1 Example

Before: `[default]\Enterprise\Site\Area\Line\InfeedCount`

After: `{Equipment Path}\InfeedCount`

If your tag path is not built with the same hierarchy as the Production Model, you can still parameterize the path using parts of the Production Model path.

Before: `[default]\Enterprise\Site\Area\Line\InfeedCount`

After: `[default]\Enterprise\Site\{Equipment Path:3}\{Equipment Path:4}\InfeedCount`

Also: `[default]\Enterprise\Site\{Equipment Path:3,4}\InfeedCount`

Here the number indicating the Area in the production model hierarchy from the enterprise level is 3, so Equipment Path: 3. Similarly Equipment Path: 4 for the Line.

7.19 Live Analysis

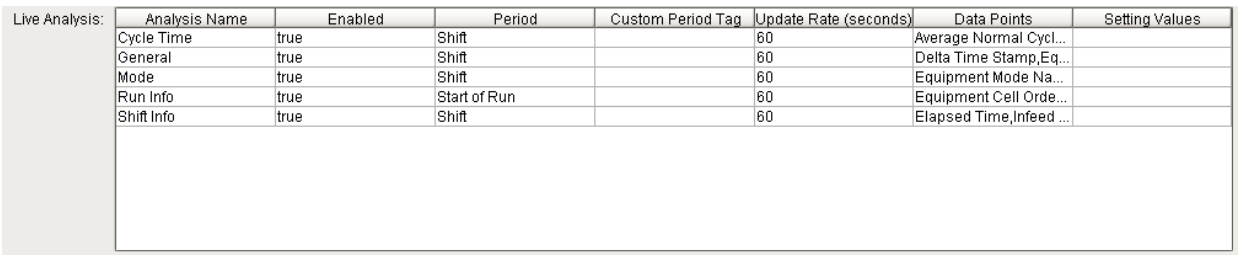
Live Analysis provides a flexible way of customizing your application to provide a set of real-time tag values that can be accessed from the Ignition designer and used in your application to provide real-time production monitoring. Live Analysis is configured in the OEE 2.0 Downtime

tab of the Production Model Designer for the Line, Cell Group and Cell production items. When a Live Analysis is created, a corresponding set of tags is created in the MES Tag Provider that provide the real-time status of those datapoints based upon the Period defined for the Live Analysis. You can create multiple Live Analysis and use those tags to drive HMI displays.

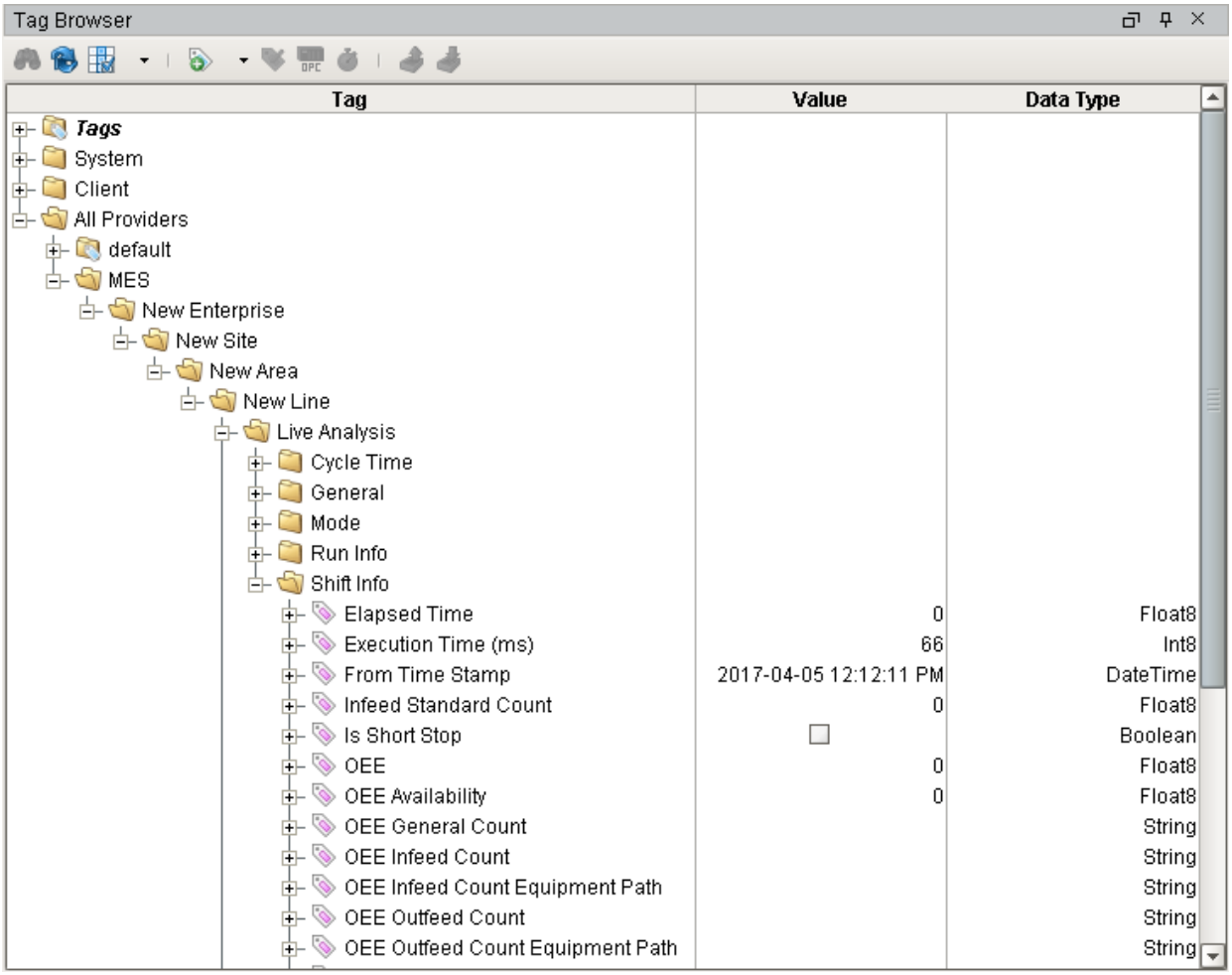
To create a new Live Analysis:

- Right click on the Live Analysis panel on the OEE 2.0 Downtime Tab in the Production Model Designer.
- Provide a Name
- Select the Period that the Live Analysis datapoints will return a value for. Valid options are Shift, Day (Midnight), Day (Production), Start of Run, Top of Hour, Custom Period Tag
- Select the frequency for how often the tag values will be updated. Default value is 60 seconds. Minimum value is 10 seconds
- Select the desired Data Points
- Add any further Settings Values required

You cannot select all Data Points in one Live Analysis. The maximum length string for Data Points is 1024 characters



Live Analysis Settings Panel in the OEE 2.0 Downtime Tab



MES Tag Provider Live Analysis Tags

7.19.1 Live Analysis Settings

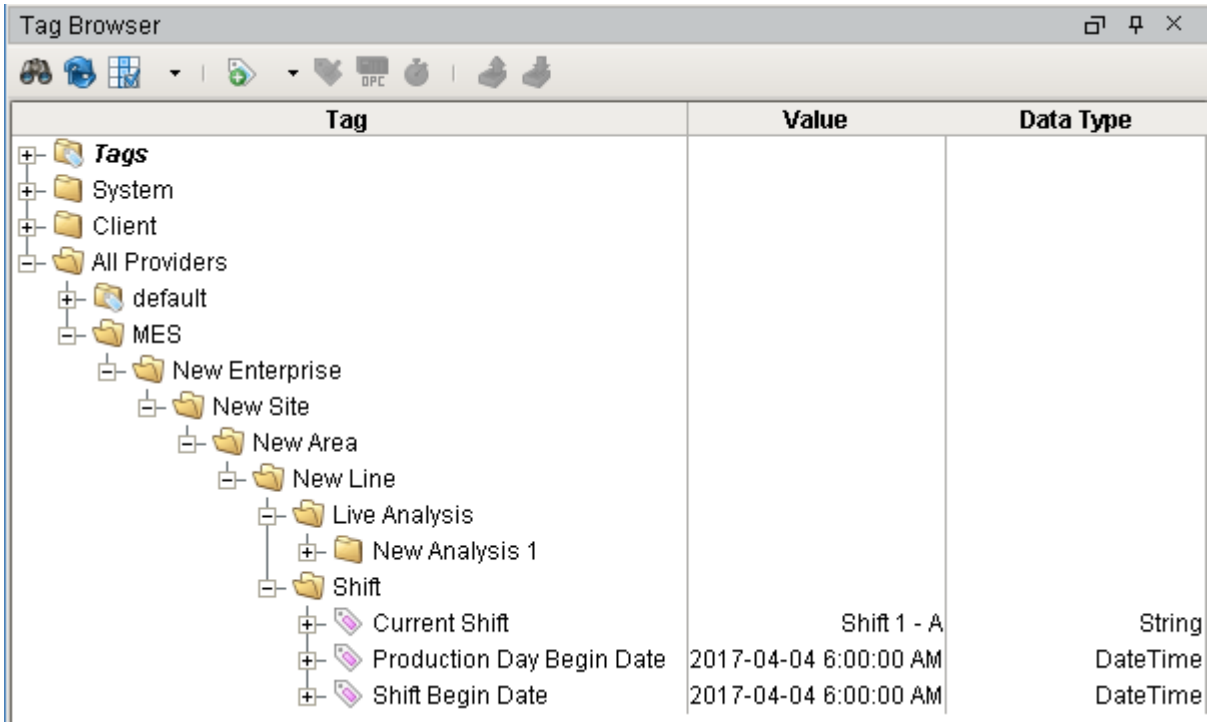
Setting	Description
Analysis Name	The name for the live analysis.
Enabled	The live analysis can be enabled or disabled with this setting.
Period	The duration of analysis can be set by **Shift**, **Day (midnight)**, **Day (production)**, **Start of Run**, **Top of Hour** or **Custom Period Tag**.
Custom Period Tag	A tag can be assigned to define the start datetime for a custom period. The end time will be the current time. It takes value in the date time data type. Example for a valid value for the custom period tag is: `2017/04/04 14:00:00`

Update Rate	The rate in seconds by which the live analysis is updated. The minimum update rate is 60 seconds.
Data Points	Data points allows you to pick and choose the values you wish to access through tags. See the table below for the listing of available data points.

7.19.2 Shift Data Points

When creating a Live Analysis, the following shift data points will be automatically created.

Data Point	Data Type	Description
Current Shift	String	The currently running shift as defined in the Ignition Schedule Management component or passed from the Shift Tag Collector path
Production Day Begin Date	DateTime	Production start time
Shift Begin Date	DateTime	Start time of the current shift



Analysis Data Points and Settings are used by Live Analysis, the MES Analysis Selector and MES Analysis Controller components, the MES Analysis Data Source for reporting and the MES Analysis Settings object.

Equipment Data Points

Data Point	Data Type	Description
Equipment		
Equipment Cell Order	Int4	Integer value that determines the cell order of the equipment within the line. Is set to *null* for the line. Is set to 0 for first cell within each cell group
Equipment Name	String	Name of the equipment as defined in the production model
Equipment Note	String	Any note that has been recorded for this piece of equipment through the Note tag collector path in the Production model will be exposed here.

Data Point	Data Type	Description
Equipment Operation Begin	DateTime	Start Date time of the currently running operation on this equipment
Equipment Operation Sequence	Int4	The ordinal number (integer) of operation
Equipment Path	String	Production model path for this equipment
Equipment Type	String	Can be Line, Cell Group or Cell
Execution Time (ms)	Int8	Time taken to execute and update the Live Analysis. Used mainly for performance debugging
From Time Stamp	DateTime	Start Date Time of current data point results
Infeed Units	String	See Infeed Units for more details
Is Key Cell	Boolean	See Key Reason for more details
Operation UUID	String	Unique Identifier for currently running operation
Outfeed Units	String	See Outfeed Units for more details
Product Code	String	Product code currently being processed on this equipment
Rate Period	String	See Rate Period for more details
Reject Units	String	See Reject Units for more details

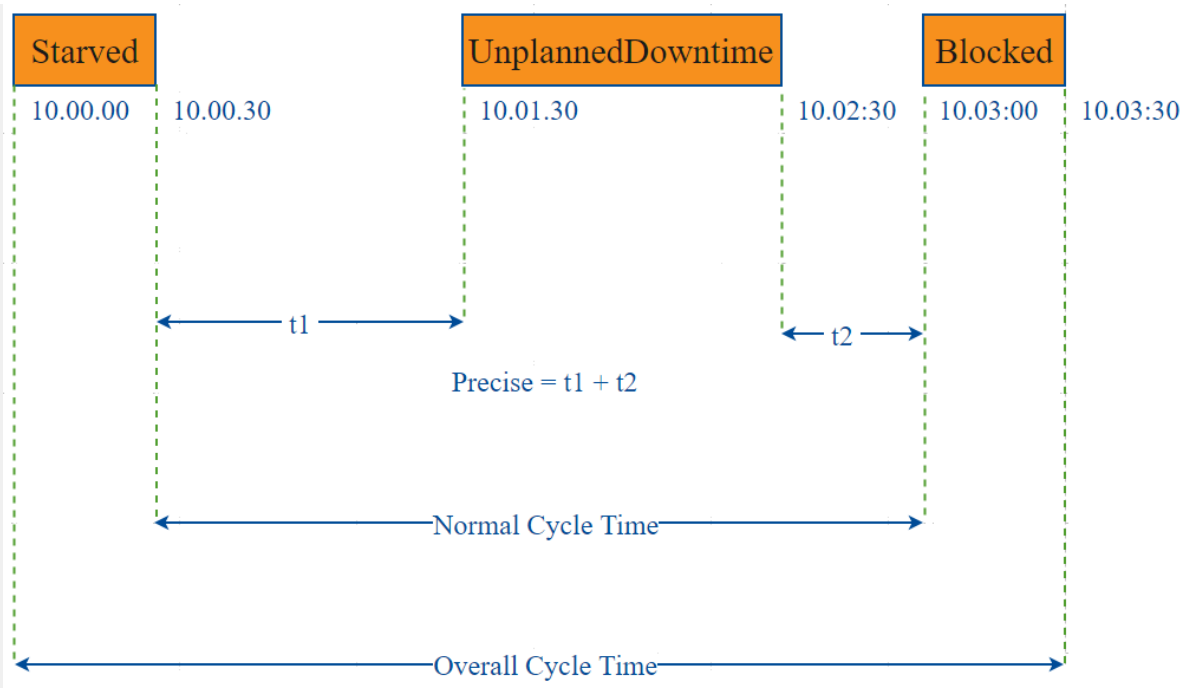
Data Point	Data Type	Description
To Time Stamp	DateTime	End Date Time of current data point results
Work Order	String	Work order currently being processed on this equipment

Equipment Count Data Points

Data Point	Data Type	Description
Equipment\Count		*Any defined counters for the production item will also appear in this folder
Equipment Infeed Scale	Float8	See Infeed Count Scale for more details
Equipment Package Count	Float8	See Package Count for more details
Equipment Reject Scale	Float8	See Reject Count Scale for more details
Outfeed-Material Out	String	Value of the default MES Counter used for OEE outfeed count

Equipment Cycle Time Data Points

The Cycle Time data points provides a number of metrics that can be used to measure the amount of time required to produce one piece. It is often used to gain an understanding of variations in production. Live Analysis provides Target, Normal, Overall and Precise Cycle Time metrics.



Data Point	Data Type	Description
Equipment\Cycle Time		
Relative Cycle Count	String	Relative Cycle Count is how many occurred for the compare by.
Target Cycle Time	Float8	Also known as Takt time, it is how often a piece must be produced to meet customer demand. It is often used to pace a production line, and it is a calculated number in seconds.
Total Cycle Count	String	Total Cycle Count is accumulative, it is sum total of all the cycle count.
Equipment\Cycle Time\Normal		**Normal Cycle Time is the actual cycle ignoring the equipment states like starved, blocked, etc.**
Average Normal Cycle Time	Float8	Average Normal cycle time in seconds for the time period selected

Data Point	Data Type	Description
Max Normal Cycle Time	Float8	Max Normal cycle time in seconds for the time period selected
Min Normal Cycle Time	Float8	Min Normal cycle time in seconds for the time period selected
Normal Cycle Time	Float8	Normal Cycle Time in seconds is the actual cycle ignoring the equipment states like starved, blocked, etc.
Equipment\Cycle Time\Overall		**Overall Cycle Time is the cycle including states like downtime, starved, blocked, etc.**
Average Overall Cycle Time	Float8	Average Overall cycle time in seconds for the time period selected
Max Overall Cycle Time	Float8	Max Overall cycle time in seconds for the time period selected
Min Overall Cycle Time	Float8	Min Overall cycle time in seconds for the time period selected
Overall Cycle Time	Float8	Overall Cycle Time in seconds is the cycle including states like downtime, starved, blocked, etc.
Equipment\Cycle Time\Precise		**Precise Cycle Time is the cycle time ignoring all the equipment states**
Average Precise Cycle Time	Float8	Average Precise cycle time in seconds for the time period selected
Max Precise Cycle Time	Float8	Max Precise cycle time in seconds for the time period selected
Min Precise Cycle Time	Float8	Min Precise cycle time in seconds for the time period selected

Data Point	Data Type	Description
Precise Cycle Time	Float8	Precise cycle time in seconds excluding states like planned downtime, unplanned downtime, starved and blocked.

Line Data Points

The Line Data points returns data for the line regardless of the Equipment the Live Analysis has been set up for.

Data Point	Data Type	Description
Line /Downtime		
Line Downtime Equipment Name	String	Name of the equipment that is responsible for causing line downtime
Line Downtime Equipment Path	String	Production model equipment path for equipment that is responsible for causing line downtime
Line Downtime Event Sequence	Int4	Every downtime event on the line is provided with an incrementing sequence number
Line Downtime Note	String	Note entered at the Line level
Line Downtime Occurrence Count	Int4	Number of downtime events for the selected period.

Data Point	Data Type	Description
Line Downtime Reason	String	The line or cell group (sub line) downtime reason. 1. When the line is down the Line Downtime Reason is the same as the Line State Name. 2. When the line is up the Line Downtime Reason is blank.
Line Downtime Reason Path	String	The full reason name for line or cell group (sub line) downtime reason. Line State name including State Class i. e. *Default/Cell Faulted*
Line Downtime Reason Split	Boolean	The line downtime reason split indicator. True is current downtime event has been split into multiple downtime events
Line Downtime State Time Stamp	DateTime	The time stamp for the equipment state change of the cell group (sub line) or cell that caused the line down time even.
Line /Meantime		
Line MTBF	Float8	The calculated Meantime (minutes) Between Failure for the selected period. Refer to Setting Up Equipment States - Meantime Metrics for more details.
Line Meantime Metrics Enabled	Boolean	Returns if Meantime metrics have been enabled for this equipment.
Line/Schedule		

Line Schedule Available	Boolean	True if this operation was scheduled
Line Schedule Available Time	Float8	Time in minutes for available production time adjusted for line schedule availability and mode.
Line Standard Count	String	Amount of product that should have been produced based on the line schedule available time and line standard rate
Line Standard Count Variance	String	Variance between standard count and actual count
Line Target Count	String	Amount of product that should have been produced based on the line schedule available time and line schedule rate
Line Target Count Variance	String	Variance between line scheduled count and line OEE outfeed count.
Schedule Rate	Float8	See Schedule Rate for more details
Line/State		
Line State Duration	Float8	The line or cell group (sub line) downtime event duration in minutes.
Line State Event Begin	DateTime	The line or cell group (sub line) downtime event begin date time.
Line State Event End	DateTime	The line or cell group (sub line) downtime event end date time.
Line State Event Sequence	Int4	The equipment state event sequence number.

Data Point	Data Type	Description
Line State Name	String	The line or cell group (sub line) state. 1. When the line is down the Line Downtime Reason is the same as the state name. 2. When the line is up the Line Downtime Reason is blank.
Line State Override Scope	String	The state override scope for a line or cell group (sub line). See Setting Up Equipment - Override Scope for more details
Line State Override Type	String	The state override type for a line or cell group (sub line). See Setting Up Equipment - Override for more details
Line State Type	String	The line or cell group (sub line) state type. See Setting Up Equipment - State Type for more details
Line State Value	Int4	The line or cell group (sub line) downtime state code. See Setting Up Equipment - State Code for more details

Equipment Mode & State Data Points

Data Point	Data Type	Description
Equipment /Mode		
Equipment Mode Name	String	Name of the current mode. See Setting Up Equipment Modes for more details
Equipment Mode Type	String	Name of the current mode type. See Setting Up Equipment Modes for more details

Data Point	Data Type	Description
Equipment Mode Value	Int4	Name of the current mode. See Setting Up Equipment Modes for more details
Mode Begin Time	DateTime	Start time of the current mode
Mode Duration	Float8	Duration of the current mode in minutes
Mode End Time	DateTime	End time of the current mode
OEE Enabled	Boolean	See Setting Up Equipment Modes - OEE Enabled for more details
Production Counts Enabled	Boolean	See Setting Up Equipment Modes - OEE Enabled for more details
Equipment /State		
Equipment Original State Value	Int4	The original value of equipment state tag collector before it is updated by using MES Value Editor component or scripting
Equipment State Name	String	Current state name
Equipment State Path	String	Production model equipment path for equipment that is responsible for causing line downtime
Equipment State Split	Boolean	True is current downtime event has been split into multiple downtime events
Equipment State Type	String	See Setting Up Equipment - State Type for more details

Data Point	Data Type	Description
State Begin Time	DateTime	Start time of the current state
State Duration	Float8	Duration of current state in minutes
State End Time	DateTime	End time of the current state

Equipment Meantime Data Points

Data Point	Data Types	Description
Equipment/Meantime		
Equipment MTBF	Float8	The Mean Time (minutes) Between Failure for the selected period
Equipment Meantime Metrics Enabled	Boolean	True if Equipment Meantime Metrics are enabled for the current equipment state

Equipment General Data Points

Data Point	Data Types	Description
Equipment /General		
Delta Time Stamp	Float8	Time gap (minutes) between the rows of data.
From Time Stamp	DateTime	Start time (minutes) of the current period

Data Point	Data Types	Description
Shift	String	Name of the current shift as set by the Ignition Schedule Management component and defined for the current line or by the value passed in the equipment shift tag collector
Shift Day Text	String	Name of the current day
Shift Day of Month	Int4	Int value of the current month
Shift Day of Week	Int4	Int value of the current day of the week
Shift Day of Year	Int4	Int value of the current day of the year
Shift ISO Week of Year	Int4	Int value of the ISO week of the year
Shift Month Text	String	Name of the current month
Shift Month of Year	Int4	Int value of the current month of the year
Shift Start Date	DateTime	Start time of the current shift
Shift Week of Month	Int4	Int value of the current week of the month
Shift Week of Year	Int4	Int value of the current week of the year

Data Point	Data Types	Description
Shift Year	Int4	Int value of the current year
To Time Stamp	DateTime	Endtime (minutes) of the current period

Equipment OEE Data Points

Data Point	Data Type	Description
Equipment/OEE		
Elapsed Time	Float8	Elapsed Time of current operation
OEE	Float8	OEE value for selected period
OEE General Count	Long	Any count value other than infeed, outfeed, reject and waste value for the selected time period
OEE Infeed Count	Long	Equipment infeed count value for the selected period
OEE Infeed Count Equipment Path	String	Infeed count tag collector path
OEE Outfeed Count	Long	Equipment outfeed count value for the selected period
OEE Outfeed Count Equipment Path	String	Outfeed count tag collector path
OEE Reject Count	Long	Equipment reject count value for the selected period
Planned Downtime	Float8	Planned Downtime duration (Double) for selected period
Runtime	Float8	Planned Downtime duration (Double) for selected period

Short Stop Time	Float8	Short stop duration (Double) for selected period
Standard Rate	Float8	See standard rate for more details
Target Changeover Time	Float8	Amount of time in minutes set for Target Changeover. See Changeover Duration for more details
Unplanned Downtime	Float8	Unplanned Downtime duration (Double) for selected period
Equipment/OEE /Availability		
Is Short Stop	Boolean	True if current equipment state is consider a shortstop. See Short Stop Threshold for more details
OEE Availability	Float8	OEE Availability value for selected period
Equipment/OEE /Performance		
OEE Performance	Float8	OEE Performance value for selected period
Infeed Standard Count	Float8	Calculated expected infeed based on standard rate
Equipment/OEE /Quality		
OEE Quality	Float8	OEE Quality value for selected period

Setting Values

The analysis results that are returned can be modified through the use of settings. Setting values provide a number of keywords as listed below.

Format for entering the keywords is k*eyword1=True, keyword2=100.0, keyword3=10.*

⚠ Settings like Enable Totalized Mode, Include Future, Last Values and Rollup Time span is meant for analysis selector and not for live analysis

Setting	Description	Use	Example
Date Format	Date format fields can be customized with this setting e.g. 'YYYY/MM/dd hh:mm:ss a'	All	Date Format = `2017/04/12 19:45:30`
Enable Totalized Mode	This setting accumulates the count. Useful for charts where you wish to display the accumulated production count over time	Not valid for Live Analysis	Enable Totalized Mode = True
Include Future	Allows for count values to be calculated in the future. Useful for charts where you want to display target counts for future runs	Not valid for Live Analysis	Include Future = True
Last Values	Only the latest values are shown.	Not valid for Live Analysis	Last Values = True
OEE Availability Cap	The maximum value calculated can be capped with this setting	All	OEE Availability Cap = 100.0
OEE Performance Cap	The maximum value calculated can be capped with this setting	All	OEE Performance Cap = 100.0
OEE Quality Cap	The maximum value calculated can be capped with this setting	All	OEE Quality Cap = 100.0
Rollup Time Span	If the time (seconds) between downtime events is less than the rollup time and it is the same equipment and reason,	Not valid for Live Analysis	Rollup Time Span = 30

Setting	Description	Use	Example
	then it will rollup the event into one row in the results and will increase the occurrence count.		
Row Limit	The analysis can be limited to a certain number of rows.	All	Row Limit = 10

7.20 Analysis Data Points and Settings

Analysis Data Points and Settings are used by Live Analysis, the MES Analysis Selector and MES Analysis Controller components, the MES Analysis Data Source for reporting and the MES Analysis Settings object.

7.20.1 Equipment Data Points

Data Point	Data Type	Description
Equipment		
Equipment Cell Order	Int4	Integer value that determines the cell order of the equipment within the line. Is set to *null* for the line. Is set to 0 for first cell within each cell group
Equipment Name	String	Name of the equipment as defined in the production model
Equipment Note	String	Any note that has been recorded for this piece of equipment through the Note tag collector path in the Production model will be exposed here.
Equipment Operation Begin	DateTime	Start Date time of the currently running operation on this equipment

Data Point	Data Type	Description
Equipment Operation Sequence	Int4	The ordinal number (integer) of operation
Equipment Path	String	Production model path for this equipment
Equipment Type	String	Can be Line, Cell Group or Cell
Execution Time (ms)	Int8	Time taken to execute and update the Live Analysis. Used mainly for performance debugging
From Time Stamp	DateTime	Start Date Time of current data point results
Infeed Units	String	See Infeed Units for more details
Is Key Cell	Boolean	See Key Reason for more details
Operation UUID	String	Unique Identifier for currently running operation
Outfeed Units	String	See Outfeed Units for more details
Product Code	String	Product code currently being processed on this equipment
Rate Period	String	See Rate Period for more details
Reject Units	String	See Reject Units for more details
To Time Stamp	DateTime	End Date Time of current data point results

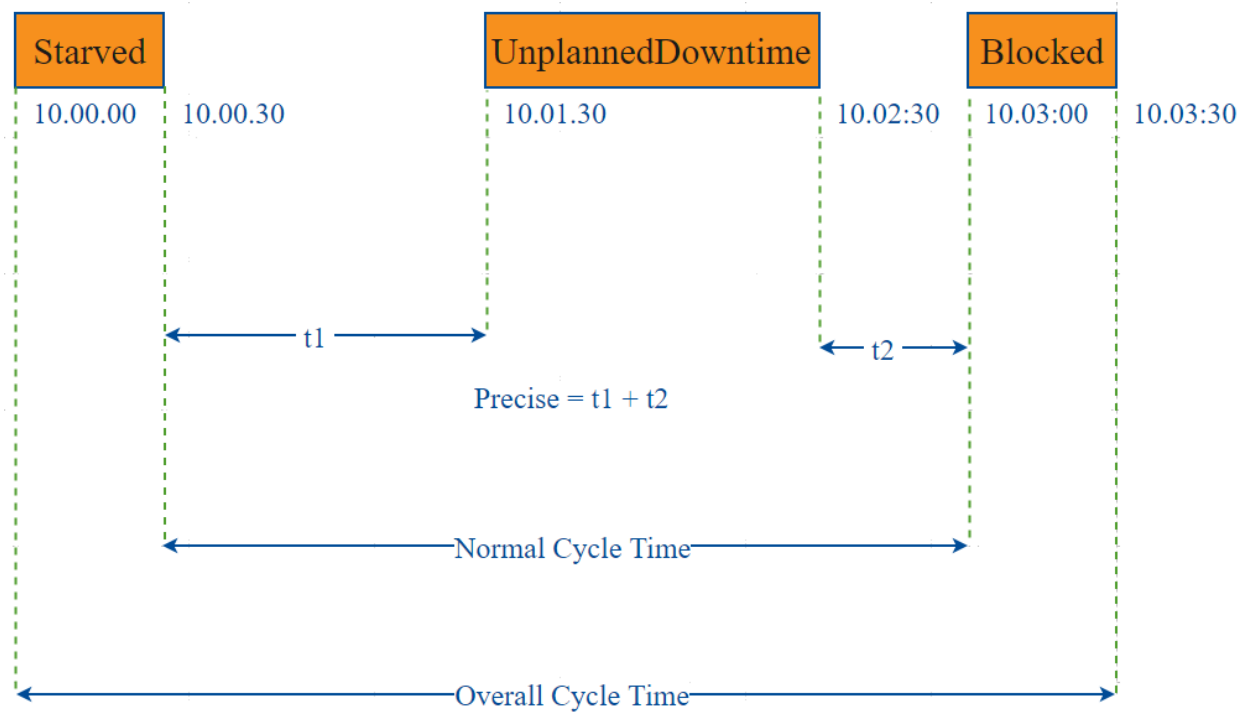
Data Point	Data Type	Description
Work Order	String	Work order currently being processed on this equipment

7.20.2 Equipment Count Data Points

Data Point	Data Type	Description
Equipment\Count		*Any defined counters for the production item will also appear in this folder
Equipment Infeed Scale	Float8	See Infeed Count Scale for more details
Equipment Package Count	Float8	See Package Count for more details
Equipment Reject Scale	Float8	See Reject Count Scale for more details
Outfeed-Material Out	String	Value of the default MES Counter used for OEE outfeed count

7.20.3 Equipment Cycle Time Data Points

The Cycle Time data points provides a number of metrics that can be used to measure the amount of time required to produce one piece. It is often used to gain an understanding of variations in production. Live Analysis provides Target, Normal, Overall and Precise Cycle Time metrics.



Data Point	Data Type	Description
Equipment\Cycle Time		
Relative Cycle Count	String	Relative Cycle Count is how many occurred for the compare by.
Target Cycle Time	Float8	Also known as Takt time, it is how often a piece must be produced to meet customer demand. It is often used to pace a production line, and it is a calculated number in seconds.
Total Cycle Count	String	Total Cycle Count is accumulative, it is sum total of all the cycle count.
Equipment\Cycle Time\Normal		**Normal Cycle Time is the actual cycle ignoring the equipment states like starved, blocked, etc.**
Average Normal Cycle Time	Float8	Average Normal cycle time in seconds for the time period selected

Data Point	Data Type	Description
Max Normal Cycle Time	Float8	Max Normal cycle time in seconds for the time period selected
Min Normal Cycle Time	Float8	Min Normal cycle time in seconds for the time period selected
Normal Cycle Time	Float8	Normal Cycle Time in seconds is the actual cycle ignoring the equipment states like starved, blocked, etc.
Equipment\Cycle Time\Overall		**Overall Cycle Time is the cycle including states like downtime, starved, blocked, etc.**
Average Overall Cycle Time	Float8	Average Overall cycle time in seconds for the time period selected
Max Overall Cycle Time	Float8	Max Overall cycle time in seconds for the time period selected
Min Overall Cycle Time	Float8	Min Overall cycle time in seconds for the time period selected
Overall Cycle Time	Float8	Overall Cycle Time in seconds is the cycle including states like downtime, starved, blocked, etc.
Equipment\Cycle Time\Precise		**Precise Cycle Time is the cycle time ignoring all the equipment states**
Average Precise Cycle Time	Float8	Average Precise cycle time in seconds for the time period selected
Max Precise Cycle Time	Float8	Max Precise cycle time in seconds for the time period selected
Min Precise Cycle Time	Float8	Min Precise cycle time in seconds for the time period selected

Data Point	Data Type	Description
Precise Cycle Time	Float8	Precise cycle time in seconds excluding states like planned downtime, unplanned downtime, starved and blocked.

7.20.4 Line Data Points

The Line Data points returns data for the line regardless of the Equipment the Live Analysis has been set up for.

Data Point	Data Type	Description
Line/Downtime		
Line Downtime Equipment Name	String	Name of the equipment that is responsible for causing line downtime
Line Downtime Equipment Path	String	Production model equipment path for equipment that is responsible for causing line downtime
Line Downtime Event Sequence	Int4	Every downtime event on the line is provided with an incrementing sequence number
Line Downtime Note	String	Note entered at the Line level
Line Downtime Occurrence Count	Int4	Number of downtime events for the selected period.
Line Downtime Reason	String	The line or cell group (sub line) downtime reason. 1. When the line is down the Line Downtime Reason is the same as the Line State Name.

Data Point	Data Type	Description
		2. When the line is up the Line Downtime Reason is blank.
Line Downtime Reason Path	String	The full reason name for line or cell group (sub line) downtime reason. Line State name including State Class i.e. *Default/Cell Faulted*
Line Downtime Reason Split	Boolean	The line downtime reason split indicator. True is current downtime event has been split into multiple downtime events
Line Downtime State Time Stamp	DateTime	The time stamp for the equipment state change of the cell group (sub line) or cell that caused the line down time even.
Line/Meantime		
Line MTBF	Float8	The calculated Meantime (minutes) Between Failure for the selected period. Refer to Setting Up Equipment States - Meantime Metrics for more details.
Line Meantime Metrics Enabled	Boolean	Returns if Meantime metrics have been enabled for this equipment.
Line/Schedule		
Line Schedule Available	Boolean	True if this operation was scheduled
Line Schedule Available Time	Float8	Time in minutes for available production time adjusted for line schedule availability and mode.
Line Standard Count	String	Amount of product that should have been produced based on the line schedule available time and line standard rate

Data Point	Data Type	Description
Line Standard Count Variance	String	Variance between standard count and actual count
Line Target Count	String	Amount of product that should have been produced based on the line schedule available time and line schedule rate
Line Target Count Variance	String	Variance between line scheduled count and line OEE outfeed count.
Schedule Rate	Float8	See Schedule Rate for more details
Line/State		
Line State Duration	Float8	The line or cell group (sub line) downtime event duration in minutes.
Line State Event Begin	DateTime	The line or cell group (sub line) downtime event begin date time.
Line State Event End	DateTime	The line or cell group (sub line) downtime event end date time.
Line State Event Sequence	Int4	The equipment state event sequence number.
Line State Name	String	The line or cell group (sub line) state. 1. When the line is down the Line Downtime Reason is the same as the state name. 2. When the line is up the Line Downtime Reason is blank.
	String	

Data Point	Data Type	Description
Line State Override Scope		The state override scope for a line or cell group (sub line). See Setting Up Equipment - Override Scope for more details
Line State Override Type	String	The state override type for a line or cell group (sub line). See Setting Up Equipment - Override for more details
Line State Type	String	The line or cell group (sub line) state type. See Setting Up Equipment - State Type for more details
Line State Value	Int4	The line or cell group (sub line) downtime state code. See Setting Up Equipment - State Code for more details

7.20.5 Equipment Mode & State Data Points

Data Point	Data Type	Description
Equipment /Mode		
Equipment Mode Name	String	Name of the current mode. See Setting Up Equipment Modes for more details
Equipment Mode Type	String	Name of the current mode type. See Setting Up Equipment Modes for more details
Equipment Mode Value	Int4	Name of the current mode. See Setting Up Equipment Modes for more details
Mode Begin Time	DateTime	Start time of the current mode
Mode Duration	Float8	Duration of the current mode in minutes
Mode End Time	DateTime	End time of the current mode

Data Point	Data Type	Description
OEE Enabled	Boolean	See Setting Up Equipment Modes - OEE Enabled for more details
Production Counts Enabled	Boolean	See Setting Up Equipment Modes - OEE Enabled for more details
Equipment /State		
Equipment Original State Value	Int4	The original value of equipment state tag collector before it is updated by using MES Value Editor component or scripting
Equipment State Name	String	Current state name
Equipment State Path	String	Production model equipment path for equipment that is responsible for causing line downtime
Equipment State Split	Boolean	True is current downtime event has been split into multiple downtime events
Equipment State Type	String	See Setting Up Equipment - State Type for more details
State Begin Time	DateTime	Start time of the current state
State Duration	Float8	Duration of current state in minutes
State End Time	DateTime	End time of the current state

7.20.6 Equipment Meantime Data Points

Data Point	Data Types	Description
Equipment/Meantime		
Equipment MTBF	Float8	The Mean Time (minutes) Between Failure for the selected period
Equipment Meantime Metrics Enabled	Boolean	True if Equipment Meantime Metrics are enabled for the current equipment state

7.20.7 Equipment General Data Points

Data Point	Data Types	Description
Equipment /General		
Delta Time Stamp	Float8	Time gap (minutes) between the rows of data.
From Time Stamp	DateTime	Start time (minutes) of the current period
Shift	String	Name of the current shift as set by the Ignition Schedule Management component and defined for the current line or by the value passed in the equipment shift tag collector
Shift Day Text	String	Name of the current day
Shift Day of Month	Int4	Int value of the current month
	Int4	Int value of the current day of the week

Data Point	Data Types	Description
Shift Day of Week		
Shift Day of Year	Int4	Int value of the current day of the year
Shift ISO Week of Year	Int4	Int value of the ISO week of the year
Shift Month Text	String	Name of the current month
Shift Month of Year	Int4	Int value of the current month of the year
Shift Start Date	DateTime	Start time of the current shift
Shift Week of Month	Int4	Int value of the current week of the month
Shift Week of Year	Int4	Int value of the current week of the year
Shift Year	Int4	Int value of the current year
To Time Stamp	DateTime	Endtime (minutes) of the current period

7.20.8 Equipment OEE Data Points

Data Point	Data Type	Description
Equipment/OEE		
Elapsed Time	Float8	Elapsed Time of current operation
OEE	Float8	OEE value for selected period
OEE General Count	Long	Any count value other than infeed, outfeed, reject and waste value for the selected time period
OEE Infeed Count	Long	Equipment infeed count value for the selected period
OEE Infeed Count Equipment Path	String	Infeed count tag collector path
OEE Outfeed Count	Long	Equipment outfeed count value for the selected period
OEE Outfeed Count Equipment Path	String	Outfeed count tag collector path
OEE Reject Count	Long	Equipment reject count value for the selected period
Planned Downtime	Float8	Planned Downtime duration (Double) for selected period
Runtime	Float8	Planned Downtime duration (Double) for selected period
Short Stop Time	Float8	Short stop duration (Double) for selected period
Standard Rate	Float8	See standard rate for more details
Target Changeover Time	Float8	Amount of time in minutes set for Target Changeover. See Changeover Duration for more details
Unplanned Downtime	Float8	

Data Point	Data Type	Description
		Unplanned Downtime duration (Double) for selected period
Equipment/OEE /Availability		
Is Short Stop	Boolean	True if current equipment state is consider a shortstop. See Short Stop Threshold for more details
OEE Availability	Float8	OEE Availability value for selected period
Equipment/OEE /Performance		
OEE Performance	Float8	OEE Performance value for selected period
Infeed Standard Count	Float8	Calculated expected infeed based on standard rate
Equipment/OEE /Quality		
OEE Quality	Float8	OEE Quality value for selected period

7.20.9 Setting Values

The analysis results that are returned can be modified through the use of settings. Setting values provide a number of keywords as listed below.

Format for entering the keywords is k*eyword1=True, keyword2=100.0, keyword3=10.*

> ⚠ Settings like Enable Totalized Mode, Include Future, Last Values and Rollup Time span is meant for analysis selector and not for live analysis

Setting	Description	Use	Example
Date Format	Date format fields can be customized with this setting e.g. 'YYYY/MM/dd hh:mm:ss a'	All	Date Format = `2017/04/12 19:45:30`
Enable Totalized Mode	This setting accumulates the count. Useful for charts where you wish to display the accumulated production count over time	Not valid for Live Analysis	Enable Totalized Mode = True
Include Future	Allows for count values to be calculated in the future. Useful for charts where you want to display target counts for future runs	Not valid for Live Analysis	Include Future = True
Last Values	Only the latest values are shown.	Not valid for Live Analysis	Last Values = True
OEE Availability Cap	The maximum value calculated can be capped with this setting	All	OEE Availability Cap = 100.0
OEE Performance Cap	The maximum value calculated can be capped with this setting	All	OEE Performance Cap = 100.0
OEE Quality Cap	The maximum value calculated can be capped with this setting	All	OEE Quality Cap = 100.0
Rollup Time Span	If the time (seconds) between downtime events is less than the rollup time and it is the same equipment and reason, then it will rollup the event into one row in the results and will increase the occurrence count.	Not valid for Live Analysis	Rollup Time Span = 30
Row Limit		All	Row Limit = 10

Setting	Description	Use	Example
	The analysis can be limited to a certain number of rows.		

7.21 Tag Collector Types

Recall that production values such as equipment modes, states and counts are recorded 24/7. If the recorded values need to be modified as production counts were off or the line mode was captured as Maintenance when it should have been Production, these scripting functions or the MES Value Editor component, can be used to correct the values.

The Tag Collector Types are used by the MES Value Editor component and the script functions listed below to read and modify production values recorded via tag collector paths and by the OEE engine.

Each tag collector type may have a different datatype and some tag collector types have a **key** (where there is more than one stored value for the tag collector type). Examples of these would be MES Counters, where the Tag Collector Type **Equipment Count** would have the default **Material Out** and any other user added mes counter names. Additional Factors would also use the key to distinguish between the user defined additional factors.

The Equipment State tag collector has an additonal parameter called the **Auxiliary Value**. The getTagCollectorValue() and updateTagCollectorValue() have an overloaded function to handle the auxiliary value name for this tag collector type.

7.21.1 Scripting Functions for Tag Collectors

- system.mes.addTagCollectorValue
- system.mes.addTagCollectorValues
- system.mes.getTagCollectorDeltaValue
- system.mes.getTagCollectorLastTimeStamp
- system.mes.getTagCollectorLastValue
- system.mes.getTagCollectorPreviousTimeStamp
- system.mes.getTagCollectorPreviousValue
- system.mes.getTagCollectorValue
- system.mes.getTagCollectorValues
- system.mes.removeTagCollectorValue

- system.mes.removeTagCollectorValues
- system.mes.updateTagCollectorLastValue
- system.mes.updateTagCollectorValue
- system.mes.updateTagCollectorValues

Tag Collector Type	Data Type	Key	Auxiliary Value	Description
Equipment Additional Factor	String	Name of user defined additional Factors	N/A	See Additional Factors for more details
Equipment Count	Long	Name of user defined MES counters and the default **Material Out**	N/A	Value of the MES Counter as defined in the key. See MES Counters for more details
Equipment Cycle Count	Long	N/A	N/A	See Analysis Datapoints and Settings - Cycle Count for more details
Equipment Downtime Note	String	N/A	N/A	Any downtime notes added through the tag collector or entered through the OEE Downtime Table component
Equipment Infeed Count Scale	Float8	N/A	N/A	See Infeed Count Scale for more details
Equipment Infeed Units	String	N/A	N/A	See Infeed Units for more details
Equipment Mode	Int4	N/A	N/A	See Setting Up Equipment Modes for more details

Tag Collector Type	Data Type	Key	Auxiliary Value	Description
Equipment Operation UUID	String	N/A	N/A	Unique Identifier for currently running operation
Equipment Outfeed Units	String	N/A	N/A	See Outfeed Units for more details
Equipment Package Count	Float8	N/A	N/A	See Package Count for more details
Equipment Product Code	String	N/A	N/A	Product code currently being processed on this equipment
Equipment Rate Period	String	N/A	N/A	See Rate Period for more details
Equipment Reject Count Scale	Float8	N/A	N/A	See Reject Count Scale for more details
Equipment Reject Units	Float8	N/A	N/A	See Reject Units for more details
Equipment Schedule Count	String	N/A	N/A	Amount of product that should have been produced (Target) based on the schedule rate
Equipment Schedule Duration	Float8	N/A	N/A	Duration of scheduled run
	Float8	N/A	N/A	

Tag Collector Type	Data Type	Key	Auxiliary Value	Description
Equipment Schedule Rate				See Schedule Rate for more details
Equipment Shift	String	N/A	N/A	The shift as defined in the Ignition Schedule Management component or passed to the Shift Tag Collector
Equipment Standard Rate	Float8	N/A	N/A	See standard rate for more details
Equipment State	String	N/A	EquipmentUUID	The unique identifier for the equipment
Equipment State	Int4	N/A	State	Equipment state value
Equipment State	Int4	N/A	OriginalState	The original equipment state before it was updated
Equipment State	String	N/A	DifferedToUUID	If the original EquipmentUUID is changed using the Downtime Table then the new uuid is DifferedToUUID
Equipment State	String	N/A	DifferedState	If the original state is changed using the Downtime Table then the new state is DifferedState

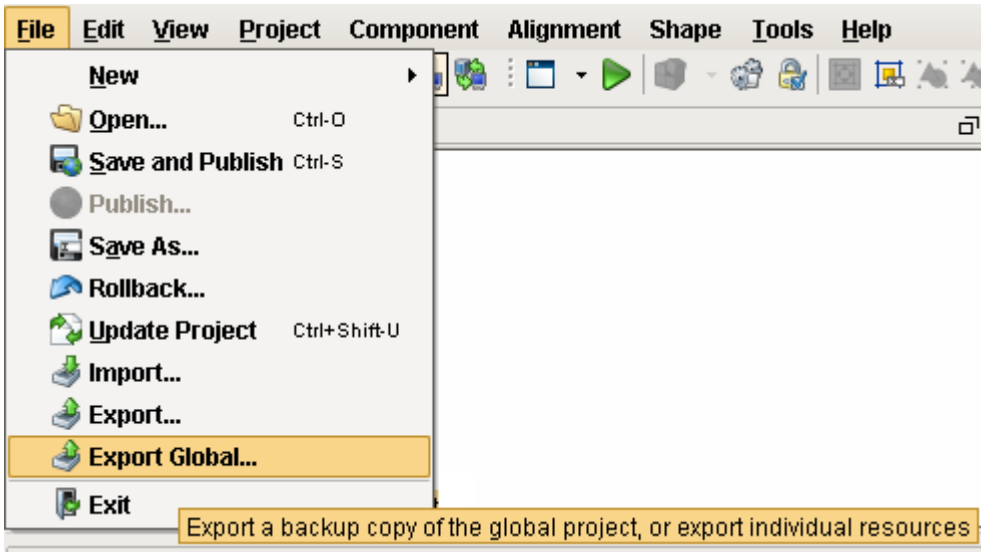
Tag Collector Type	Data Type	Key	Auxiliary Value	Description
Equipment Target Changeover Time	Float8	N/A	N/A	Amount of time in minutes set for Target Changeover. See Changeover Duration for more details
Equipment Work Order	String	N/A	N/A	Work order processed on this equipment
Line Infeed Count Equipment UUID	String	N/A	N/A	Unique identifier for the equipment where the line infeed count came from
Line Outfeed Count Equipment UUID	String	N/A	N/A	Unique identifier for the equipment where the line outfeed count came from

7.22 Exporting the Production Model

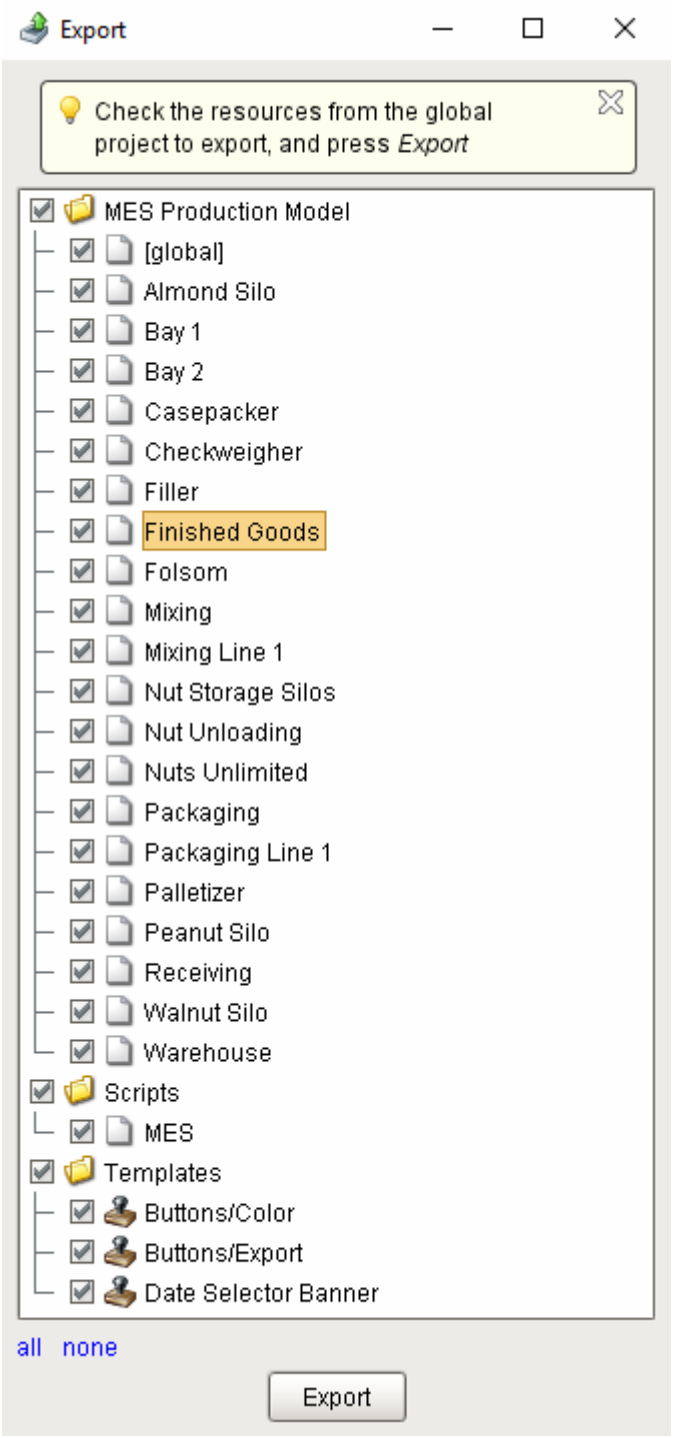
Every production item in the production model (site, area, line, cell group, cell, location) is considered by Ignition as a separate global resource. This allows multiple developers to be working on different production item configurations at the same time as the Ignition lock mechanism is by global resource.

7.22.1 Export Global

Ignition provides the ability to export global resources using the **File > Export Global** menu item.

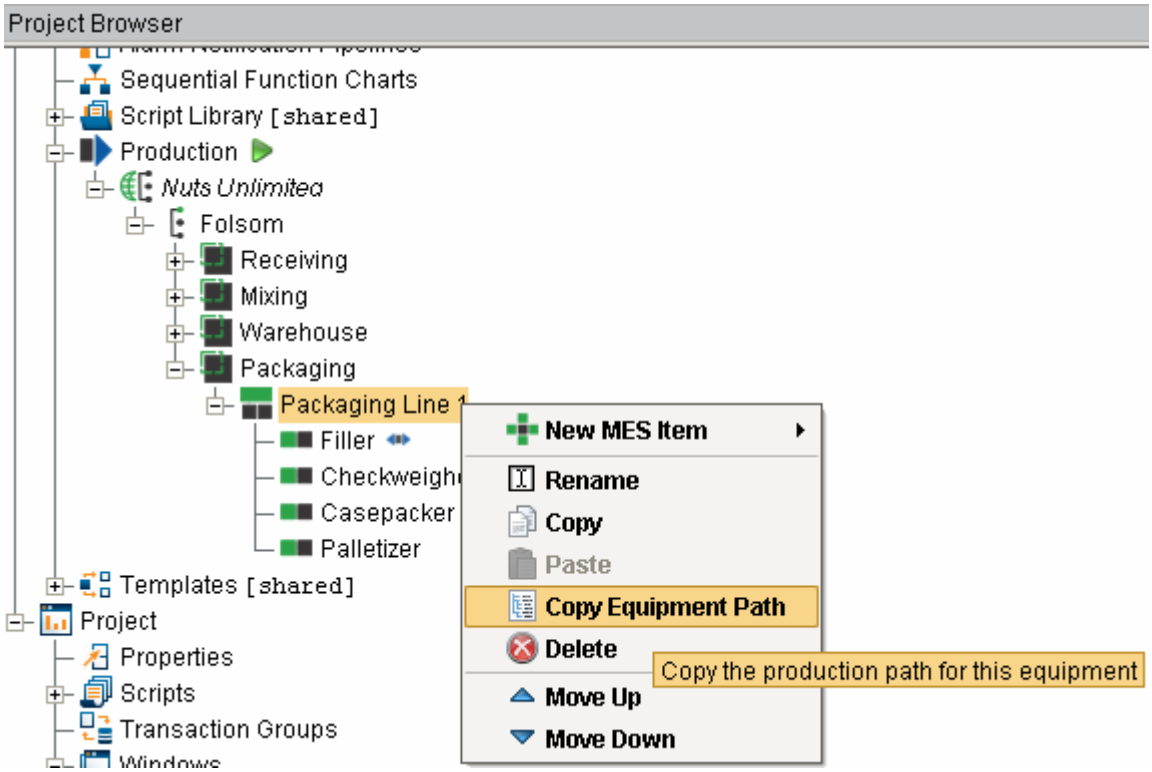


The MES Production Model export screen flattens out the production model which can make it difficult to select certain items particularly if different production items have the same name, however this method works well when you need to export the entire production model from one gateway to another.



7.22.2 Copy and Paste

When you want to make a copy of a production item with associated production items beneath it, you can right-click on the production item and select **Copy**, then select the production item one level up from the copied production item, right-click and select **Paste.** This works at the site, area, line, cell group, cell and location level, and also works when making a copy in the same gateway or from one gateway to another gateway.



7.22.3 Parameterized Tag Paths

When you have an implementation that has many identical production lines, the ability to copy and paste along with Parameterized Tag Paths provides a fast method for duplicating lines with dynamic binding to the tag source of data. By building a standard interface between the PLC logic and ignition tag structure, you can configure once and rollout to many lines.

For more information, refer to the Parameterized Tag Paths help.

8 MES Modules

8.1 OEE 2.0 Module

New to OEE?

Download and test drive this most powerful MES solution available anywhere!

Download and Install Module

A Simple Workflow

Step 1. Database Connection

Step 2. Configuring MES Databases

Step 3. Installing the Production Simulator

Step 4. Production Model Configuration

OEE Module in a nutshell

Click on the topic you would like to learn more about ...

OEE Downtime

- Components
- Scripting
- Objects

Product Data Sheet

To see the Product Data Sheet,

click on **OEE Downtime Module**

Click here to see Knowledge base articles of OEE Downtime.

Info

This module uses the following scripting functions:

system.mes.oee

Read this section about licensing...

Licensing and Activation

8.1.1 What Is OEE?

OEE stands for Overall Equipment Effectiveness and is used to monitor manufacturing effectiveness. The resulting OEE number, represented as a percentage, is generic and allows comparisons across differing industries.

Efficiency is not simply the ratio of machine run time to scheduled time. Look at the situation of your manufacturing line or process running at half speed with 0 downtime. This is truly only 50% efficient. Or what if 10% of the product being produced does not meet your minimum quality and must be reworked. This equates to 90% efficient, which does not take into account the effort to rework or the losses of raw material.

There are three factors, all represented as a percentage, taken into consideration for the final OEE result:

OEE Availability

OEE Availability is the ratio between the actual run time and planned production time. The planned production time does not included breaks, lunches and other pre-arranged time a production line or process may be down.

Example: If a line is run for one 8 hour shift with two 15 minute breaks and one 30 minute lunch, then the planned production time is 7 hours (determined from 8 hours - 15 minute break - 15 minute break - 30 minute lunch). If during the production run, there are 25 downtime events totaling to 45 minutes of downtime, then the run time is 6 hours and 15 minutes (derived from 7 hours of scheduled time - 45 minutes). The OEE Availability of 89% is calculated by actual run time divided by scheduled run time, or 6 hours 15 minutes divided by 7 hours.

OEE Performance

OEE Performance is the ratio between the actual number of units started (not the number that have been produced) and the number of units that theoretically can be processed based on the standard rate. The standard rate is the rate that the equipment is designed for. Performance is not based on the number of units produced, but, on what the line was designed to process over a given period of time.

Example: If a work cell is designed to process 10 units per minute we can calculate the theoretical amount of units it can process in a given amount of time. Using the 6 hours and 15 minutes of actual run time from the above example, a total of 3750 units would be processed (or started). Calculated by taking 6 hours and 15 minutes (375 minutes) times 10 units per minute. If the actual number of units processed is 3000, then the OEE Performance is 80% (calculated by 3000 / 3750).

OEE Quality

OEE Quality is the ratio between good units produced and the total units that were started.

Example: Taking the number of units produced from above of 3000, if 200 units were rejected at the quality inspection station, then 2800 good units are produced. The OEE Quality is 93% calculated from 2800 divided by 3000.

OEE

The final calculation is OEE = Availability x Performance x Quality.

Example: Using all the numbers from above, 89% x 80% x 93% = 66%.

This may seem like a low number but it is important to kept in mind that the OEE is not to be compared to 100%. The OEE result from this production run is compared to other production runs; however, using Sepasoft's OEE Downtime and Scheduling module allows much more than just comparing OEE results between production runs. It allows you to compare OEE results between operators, viscosity, mechanics, products, raw material vendors and any user defined factor you can think of.

OEE is a well-established performance metric that takes into account Equipment Losses usually broken into the categories of Availability Loss, Performance Loss, and Quality Loss. It measures performance with respect to Planned Production Time.

Applying OEE

OEE scores may be compared across divisions, sites, assets, or products and can be used to compare production lines that produce different products and plants of different sizes in a meaningful way. Even small increments in OEE can boost the efficiency of a manufacturing plant and when combined with analytics such as SPC, will result in high performance.

Six Big Losses

To be able to better determine what is contributing to the greatest loss and so what areas should be targeted to improve the performance, these categories (Availability, Performance and Quality) have been subdivided further into what is known as the ‘Six Big Losses’ to OEE.

These are categorized as follows:

Availability	Performance	Quality
Planned Downtime	Minor Stops	Production Rejects
Breakdowns	Speed Loss	Rejects on Start up

What Is TEEP?

Where OEE represents the equipment efficiency during a production run, Total Effective Equipment Performance (TEEP) represents the equipment utilization against a calendar period. For example, 365 days a year, or 24 hours a day. It can also be thought of as asset utilization and will help in the decision making process of purchasing new equipment.

The calculation for **TEEP = Loading * OEE**.

Loading

If a production line is scheduled for 5 days, 24 hours each day, over a 7 day period, then the **loading** is 71% calculated by (5 x 24) / (7 x 24).

Example: During the same time period that was used to calculate the **Loading**, we will make up an OEE result of 82%. The actual OEE value used must be the OEE result for all production runs of the same calendar time period that were used to calculate the Loading value.

TEEP is 71% * 82% = 58%

Downtime Tracking

OEE provides a method to monitor the efficiency of your production facility and tracking downtime provides information of where to focus efforts to improve efficiency. Think of it this way, if your production line typically runs at 69% OEE, what actions do you take to increase it? OEE alone doesn't tell you what factors are preventing your efficiency from being higher than 69%.

In the simplest form, downtime tracking will identify the production cell (machine or process) that is preventing your production line from producing product. This can be done manually, but history has shown that manually collected downtime information is inaccurate. In addition, if it is manually collected on paper log sheets, then someone has to further enter the details into a program or spreadsheet to be able to organize it into actionable information used to focus your efforts to make improvements. Putting recording inaccuracies, extra labor and typos aside, by the time the information is available, it is old.

Tracking downtime automatically or semi-automatically solves the issues associated with manual tracking. In a perfect world, monitoring all downtime reasons automatically is the ideal solution. But in the real world, this can be difficult, pricey, or just not practical. For this reason, it is important for downtime tracking software to support an automatic reason detection with a manual override.

For example: if an operator presses the stop button because they see a bottle laying on its side feeding into a filler, then the only automatic reason that can be detected is "operator pressed stop button". Now the operator should be able to override this reason with more specific information.

Once the period of time that production cells were not producing product and the associated reasons are recorded, analyzing the summary of the reasons will identify where effort should be focused to improve efficiency.

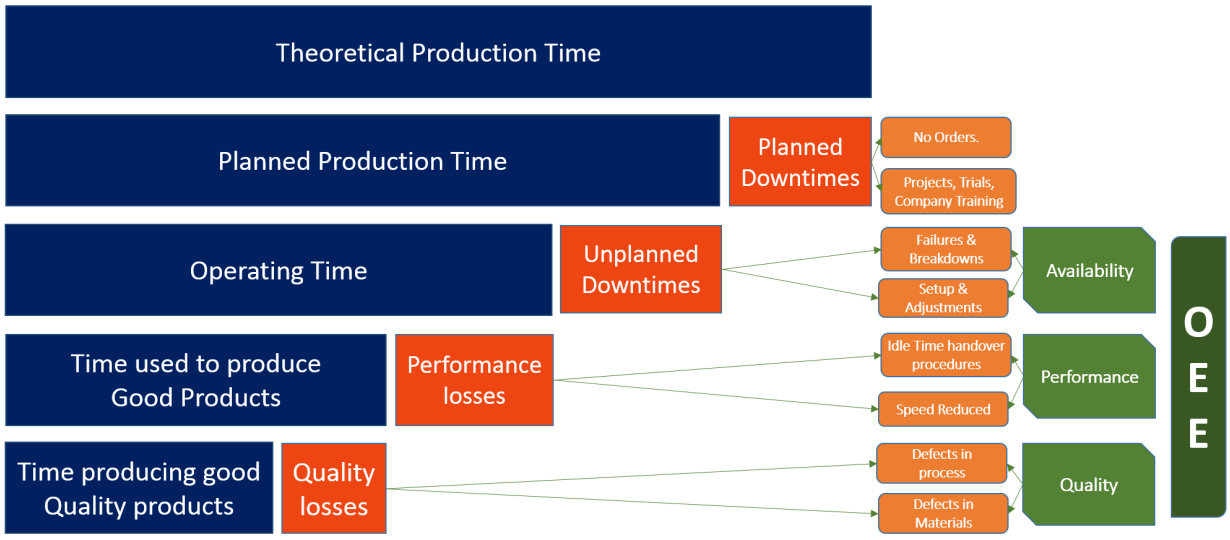
8.1.2 Features

Improving production efficiency is the key to increasing profit and reducing capital expenditure. It can make the difference competitively, however, it can also be very challenging because it requires more than just installing software. Improving efficiency requires commitment from management, maintenance, production and IT departments, as well as integration, training, actions to reduce downtime and new operational procedures. The OEE 2.0 module helps drive your continuous improvement initiatives by giving you the tools and access to data to diagnose the inefficiencies within your production.

The first step in improving efficiency is knowing where you are starting from. Think of it like improving the gas mileage of your car. You must start by determining your current gas mileage before you can begin making changes to improve your mileage. Once you know your existing OEE and have tracked the causes of downtime, then you can finish the process and start fixing the sources of your production inefficiencies.

The OEE Module combines Production Scheduling, Run Control, automated real-time line status, production counts tracking, and Overall Equipment Effectiveness (OEE) calculations to give manufacturers and operations managers a robust software package for production schedule planning and automating the measurement of operational efficiency in order to help drive continuous improvement initiatives.

It is not necessary to use all the features that we provide. We packaged them together because the combination provides the best tools for the improvement of production efficiency. If we only track downtime, we would not see the full picture as downtime only tells us if a machine is running, not if the machine is actually producing a quality product. If we only track OEE, we would know whether efficiency is lower than normal, but not why or what actions to take to improve it. Inefficiencies can also result from ineffective procedures or a lack of communications between departments. This is where the scheduling helps by providing current schedule information to all associated departments, improving communication and reducing unnecessary delays. The OEE Downtime and Scheduling Module allows you to see the whole picture, resulting in the improvement of your production in every aspect.



Real-Time Efficiency Management

OEE, downtime and production data is displayed in real time, empowering front line operators and supervisors to respond to real-world situations as they are happening today, not last week. Real-time management communication between the enterprise and plant floor helps facilities accomplish continuous improvement initiatives, as well as business strategies such as LEAN and Six Sigma. OEE calculations monitor manufacturing effectiveness and allows comparisons across plants.

Automatic and Manual Entry

Data is automatically captured from devices and recorded via OPC tags rather than relying on manual entry and interpretation by production staff. As an added benefit, valuable man hours will be reduced for data entry, and can be utilized elsewhere. Efficiency is not simply the ratio of machine run time to scheduled time. Look at the situation of your manufacturing line or process running at half speed with 0 downtime. This is truly only 50% efficient. Or what if 10% of the product being produced does not meet your minimum quality and must be reworked. This equates to 90% efficient, which does not take into account the effort to rework or the losses of raw material.

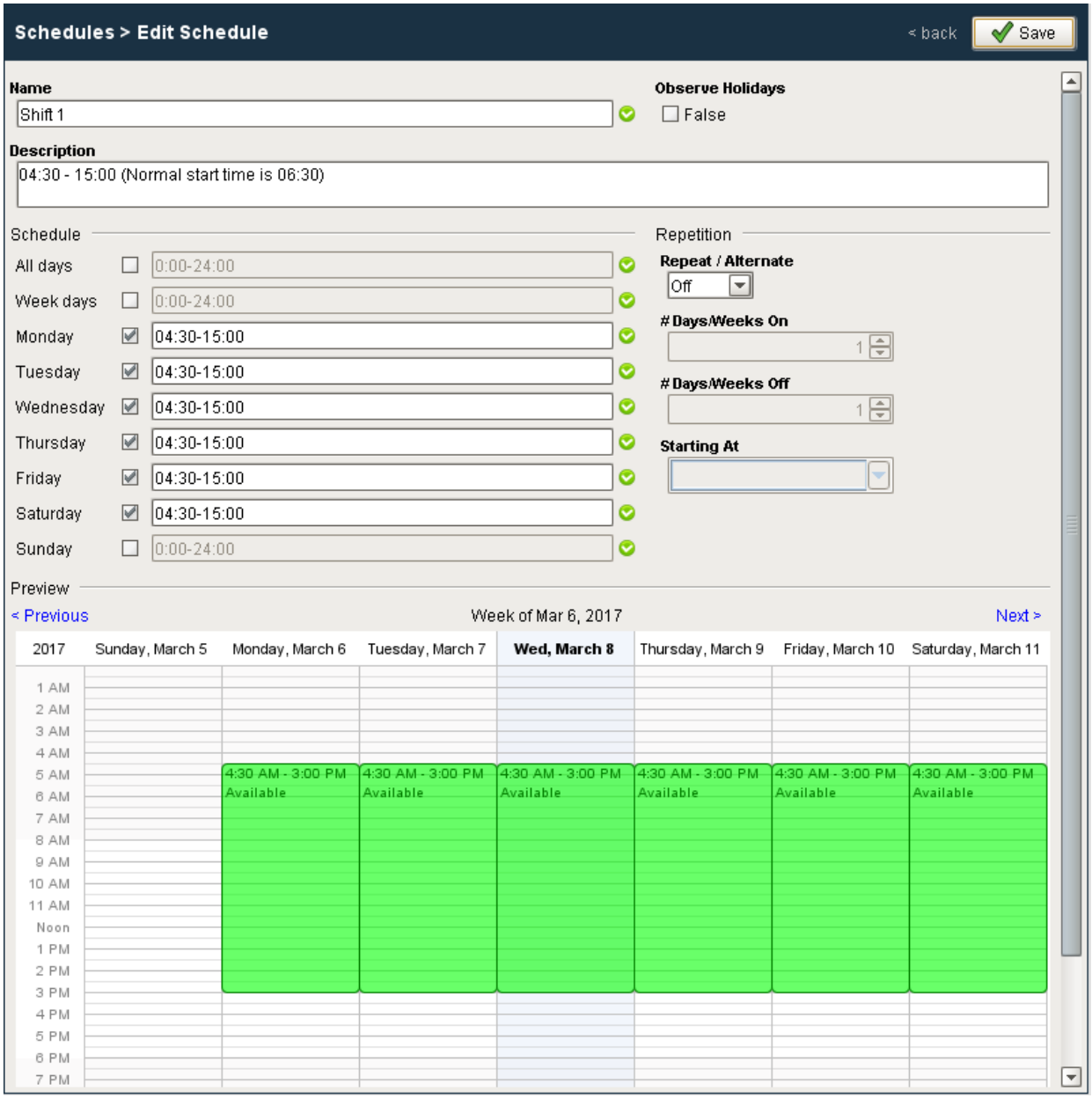
Improvements over OEE 1.0 Module

The new OEE 2.0 module adds new features and enhancement over the original OEE 1.0 module. Those new features are detailed below.

Shift Management

Identifying the current Shift can now be tied into the Ignition Shift Schedule Management System, where custom shifts can be defined for daily, weekly or rotating schedules. If the Ignition Shift Schedule Management System does not support your needs or is not required, we now provide the hooks so you can tie current production into a custom shift scheduling solution .

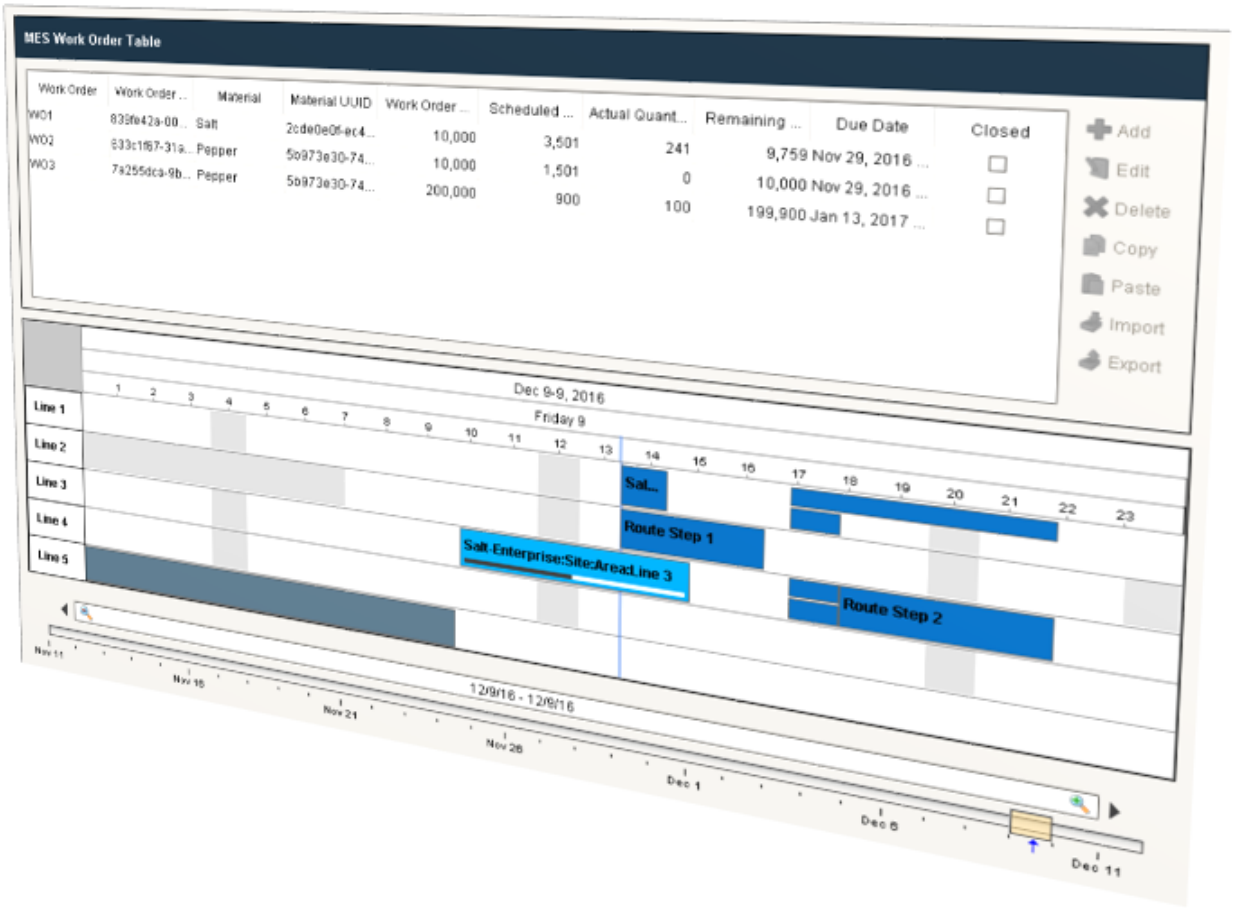
Refer to Shift Configuration for more details.



Production Scheduling

The Scheduler is now based on the same components as the Track and Trace module providing Work Order **Drag and Drop** capability, **Auto-Delay** feature, custom categories (hold schedule) and **routes**.

Refer to Detailed Production Scheduling for more details.



Equipment Modes

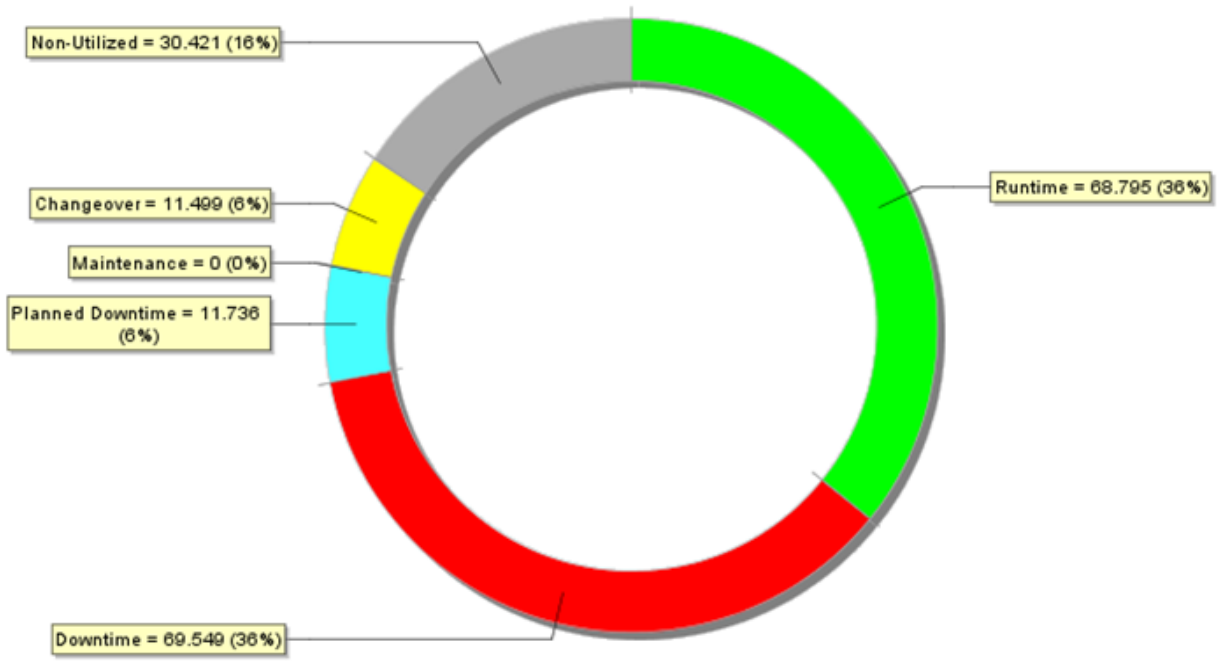
In manufacturing, it is fairly common for a production line to be running, but not actually producing parts or finished goods as part of a work order or product code run. Examples of this may be when the line is being run during scheduled maintenance to verify work has been performed to a satisfactory level. There are times when a line may be running as part of new product introduction or testing, or times when operator training is occurring. When the line is getting ready for a production run, there may be a period of time that the line is in a changeover or setup mode, and there may be times during a production run, when the line mode changes back to setup say after a fault, and material has to be run back through the extruder cell. Modes provide a more logistical / planning / scheduling view of what is being asked of the equipment, rather than the Equipment State that provides actual status of the equipment.

In OEE 2.0, it is now possible to define modes that the line or equipment may be in and you can define whether production counts should be included during these modes and used in OEE calculations. Modes allows for a more granular analysis of equipment utilization; how much time does a line spends in maintenance, training, changeover, setup, while still allowing the

equipment states of running, blocked, idle, faulted to be captured. As production counts and equipment cycles are captured in all modes, you can still use these counters to determine run hours on equipment, strokes on die sets etc., that may drive maintenance scheduling and other activities.

Default Equipment Modes are provided for IDLE, CHANGEOVER, PRODUCTION, MAINTENANCE, TRAINING, TESTING. Custom modes can also be added.

Refer to Setting Up Equipment Modes for more details.



Equipment States

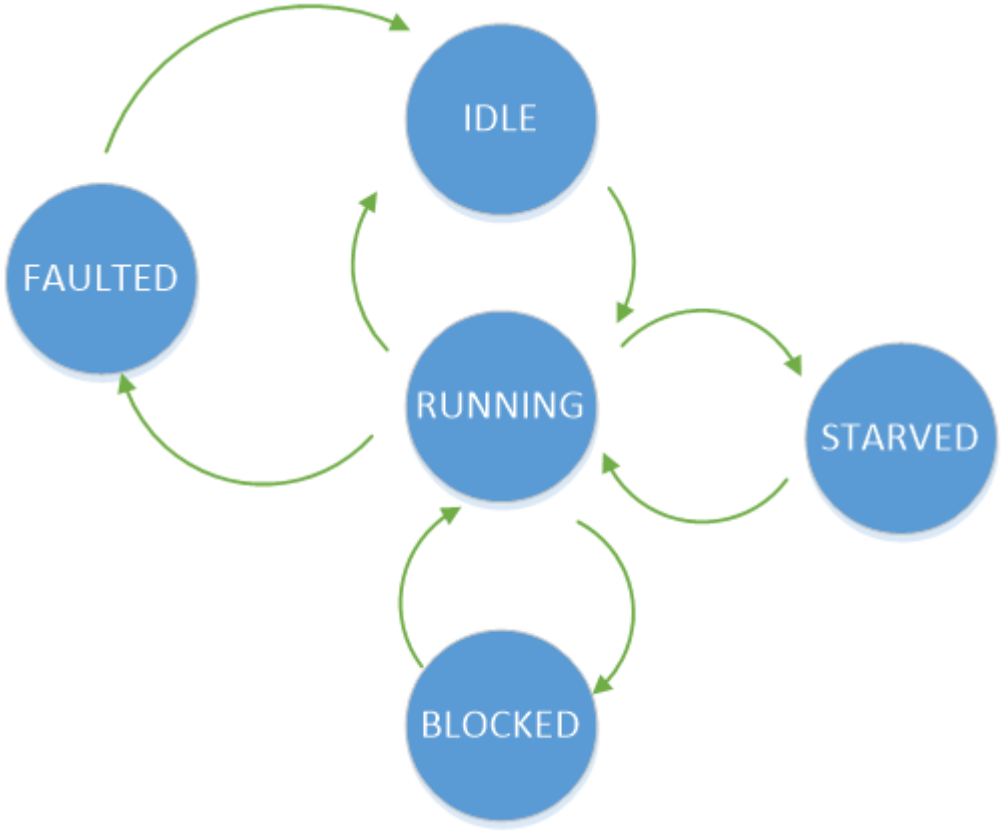
Equipment States represent the status of the line or cells within a line that provide an indication of whether the equipment is off-line, idle, running, faulted, blocked, starved, in CIP etc. It is entirely separate from the equipment mode, which provides a more logistics/scheduling view of equipment. The equipment status will generally come from a plc tag that provides real-time state information of the equipment, but can really come from any source that can populate an ignition tag. This allows for manual entry screens of operator input equipment state or data parsed from a flat file entry.

Equipment States (formerly known as Downtime events) can now be grouped, allowing multiple states to be considered as **Running** by OEE i.e. Loading, heating, molding Running. No need to force your equipment state to 1 or group a bunch of states into one value. True equipment state is now captured and stored allowing for greater cycle time analysis.

In OEE 1.0, equipment states were configured in the production model designer for each line and cell. In OEE 2.0, equipment states can be defined in the client using the Equipment Manager component, and a common set of equipment states (class) can be used across a set of lines and equipment.

Machine State can now be configured to be **Operator Selectable - REQUIRED** when you require an operator to select the actual cause or add a note for a certain event. Downtime table component will flag the events that need to be over-ridden. User roles can also be configured on who can select operator selectable reasons.

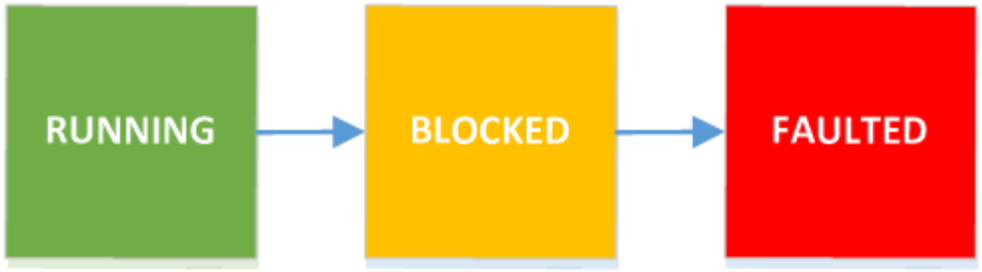
Refer to Setting Up Equipment States for more details.



Downtime Detection Modes

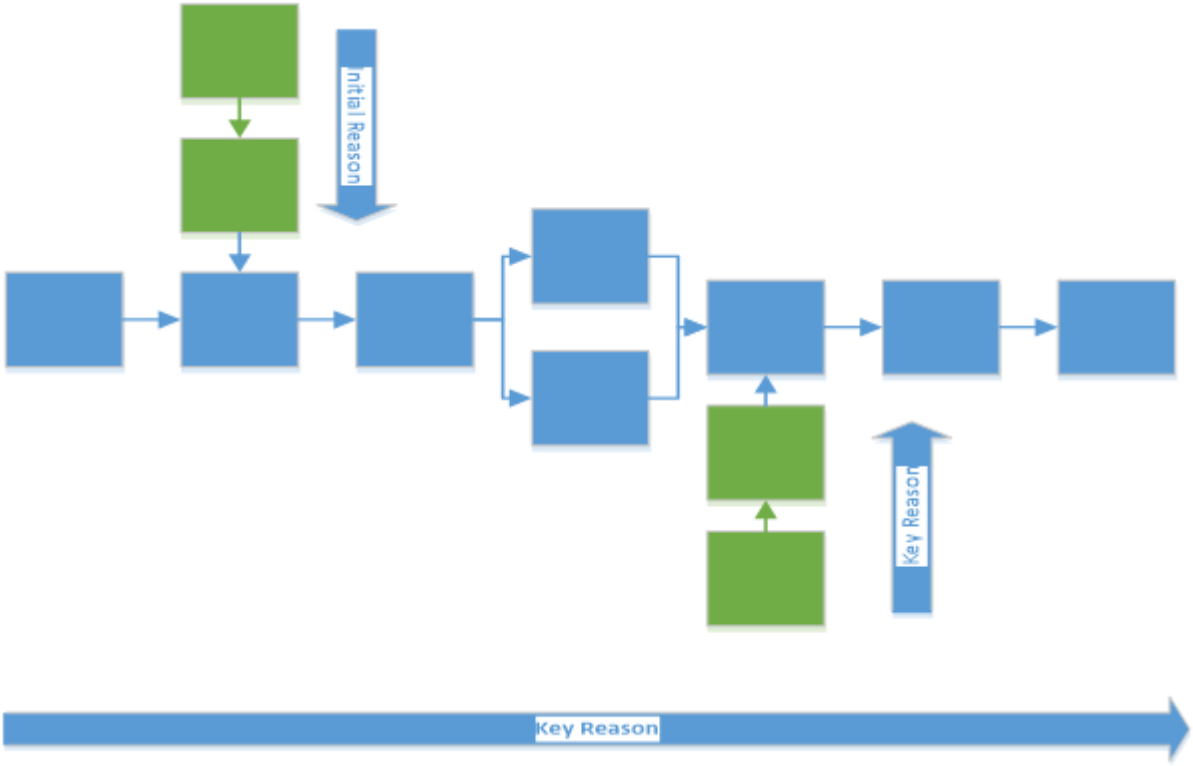
The OEE engine provides a number of ways to determine how line downtime was caused. If you have a single cell for your line, you can use Line State to have a single state tag determine if the line is running or faulted. When a line consists of multiple cells each performing a process on material, **Initial Reason** and **Key Reason** Downtime Detection Methods can be used to determine which cell is causing the line to be down. In OEE 2.0, Key Reason Detection Method now supports placing a primary cell anywhere within a line.

Refer to Downtime Detection Mode for more details.



Work-Center and Sub-Line OEE Operation

Not all manufacturing processes fit perfectly into providing a Single Line OEE Metric, such as complex work-center type operations or production lines with multiple sub-lines feeding in or out to a main line. OEE 2.0 adds support to define sub-lines with their own downtime detection method, equipment modes and states. Sub-lines can inherit their schedule from the main line, but can also run independently.



Running Changeovers, Data Capture and Runtime Monitoring

- Never stop another production run to switch over products. OEE 2.0 supports the running changeover in multiple products.
- Equipment Mode, State, counts, shift and standard rate tags are always monitored and recorded. Never lose production data again because a run wasn't scheduled.

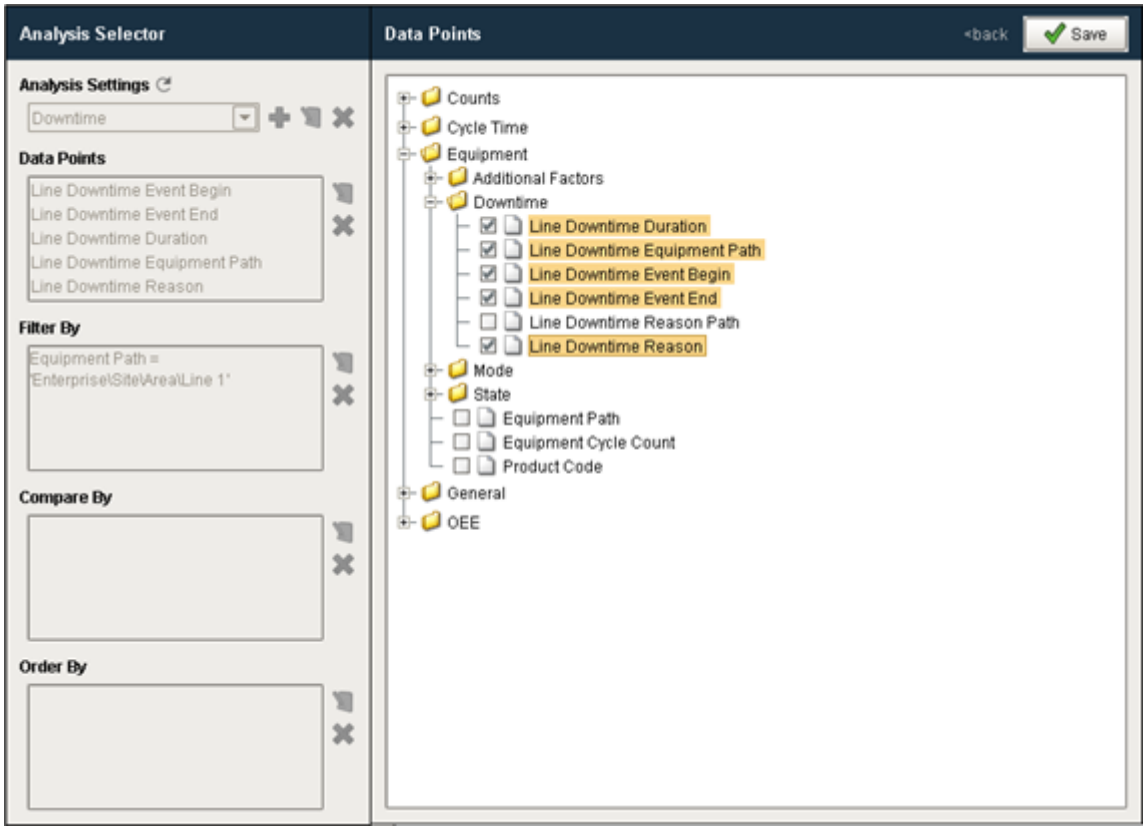
- You can now create multiple 'live' OEE values for user defined time periods. How often these values are updated is now user configurable and uses an optimized cache to provide real-time values that can be displayed for run-time monitoring. Users will now be able to see real-time OEE metrics and production counts updated as often as they want.

Image result for 7 24

Powerful New Analysis Engine

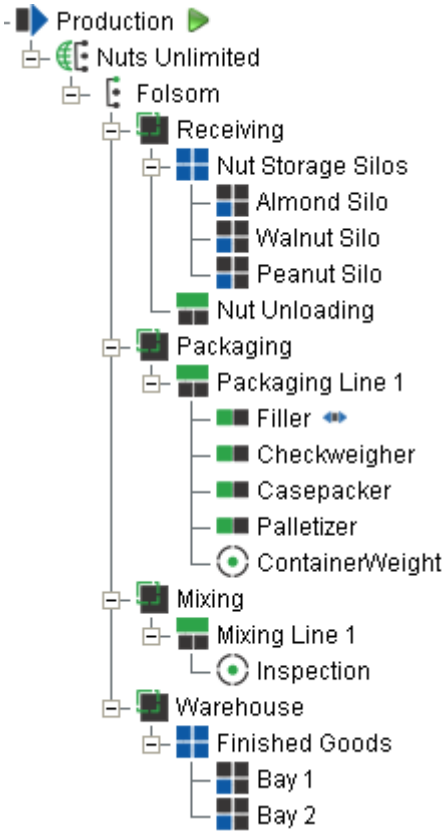
New analysis engine is simplified and optimized to provide faster, more powerful analytics aggregating data from all MES sources as well as any custom value sources. OEE analysis now occurs on-the-fly and can be shared with other systems and web reporting tools using built-in system functions. Criteria can be changed to reflect the OEE based on your selection. Stored analysis now supports user security and ability to promote reports to the public domain.

Built-in system functions now provide a scripting method for running a stored analysis and pushing that data out to a user configured database table. This can be done from a gateway timer, tag change event, etc. or through a web service call to allow the sharing of OEE analysis with other systems and Company Web Reporting tools.



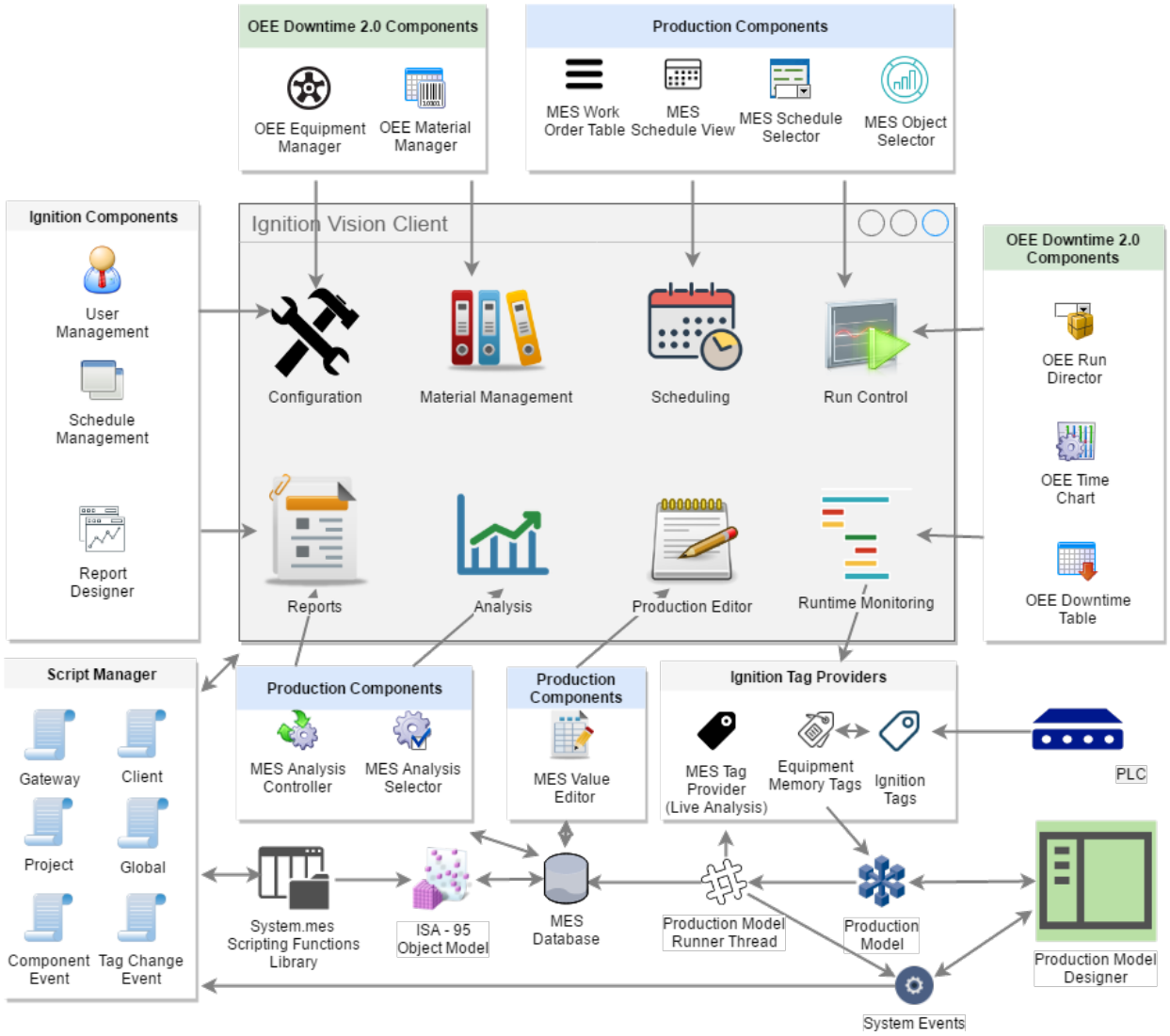
Designer Enhancements

- The production model is now saved as individual project resources for each production item, allowing for faster saves and multiple developers making changes at the same time.
- Production items can also be moved between gateways without losing their identity.
- OEE data collection continues uninterrupted even while saving the production model.
- OPC production tags are replaced by the MES Tag Provider and Live Analysis that allows you to define the real-time tag data that you need
- Configurable MES Backup options will allow you to select and transfer product codes, work orders, operations etc.



8.1.3 Framework

The OEE 2.0 Module framework shares a number of common MES components, scripting functions and objects that are also used by the Track & Trace Module. This commonality provides a seamless integration between the two modules when a project implementation requires Production Control, Lot Tracking, Inventory Management, Production Scheduling, OEE Analysis and Downtime Tracking.



OEE 2.0 Framework Diagram

The framework consists of...

- Ignition Gateway MES Settings
- Production Model
- Components
  - OEE
  - Common MES
- Scripting Functions
  - OEE
  - MES
  - Work Order

    - Analysis
- Objects
    - OEE Objects
    - MES Objects
- MES Tag Provider (Live Analysis)
- MES Database Tables

8.1.4 Equipment Configuration

Setting Up Production Lines for OEE

There are several steps that need to take place in order to configure the OEE module to be able to provide accurate production count and OEE metrics, as well as schedule Production and Maintenance Work Orders if required.

1. Model the production line(s) in the Production Model Designer
2. Add Additional Factors
3. Add Machine Modes and States in the Equipment Manager
4. Create Product Codes (Material Definitions) and add Product Code Line Configuration information such as Standard Rate in the Material Manager

In the Equipment Configuration section of the manual, we will deal with these steps.

In this section

Modelling the Production Line

How you model a production line in the Production Model will be dependent upon the following factors:

- **Type of Line** (single machine, simple contiguous process, parallel processes, shared cells across lines, complex sub-lines, primary cells, cell groups with cell groups)
- **Line Downtime Determination** (When is the line considered to be down? All cells, any cells, particular cell)
- **How production is scheduled** (if parts of a line are scheduled separately from each other, then they may be considered two separate lines)
- **Interface to cell** (if a cell has no real-time accessible interface, its state and output may be combined with an adjacent cell)

The importance of modelling the line correctly based on the above factors will drive the usability and usefulness of any production data or metrics generated. If a production line of many cells is modeled simply as a single cell, then features provided by the OEE module such as downtime detection methods, cell cycle time, aggregation of machine states by cell, and visual aspects of the OEE components such as the OEE Time chart will all be diminished. Forcing management to schedule production runs on multiple lines that they did not do prior to the OEE implementation, as well as only allowing a single scheduled run on a a set of cells where traditionally multiple schedules existed are signs that the model may not be correct.

In modelling the Line, you will create a Production Model in the Designer that

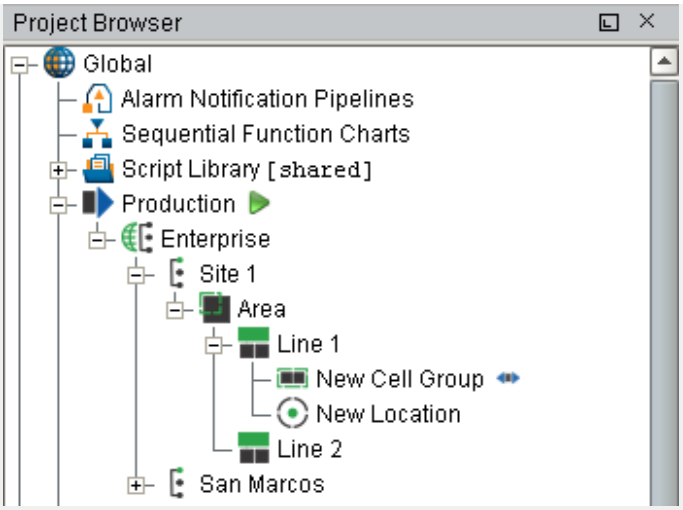
- Defines your Enterprises Sites, Areas, Lines and Cells
- Sets up Line Downtime Detection Method to define how line downtime is determined
- Connects states and counts from the equipment plc interface to the production model

What Is The Production Model?

When any of the core MES modules (OEE, SPC, Recipe, T&T) are installed, the Production Model is added to the **Global** project resources in the Project Browser window of the Ignition Designer. The Production Model allows you to define your manufacturing process in a tree view form and provides an organized way to configure, control, and analyze your manufacturing activities. It provides the foundation on which the MES modules are built.

The Production Model is a hierarchy of Sites, Areas, Lines, Cell Groups, Cells, Locations, Storage Zones and Storage Units. Typically, Lines and Cells are used to represent machinery or equipment where a process occurs transforming raw materials into sub-assemblies or finished goods. Storage Zones and Storage Units are typically used to define where to get or store material.

Lines and Cells defined in the production model should be considered to be equipment that is bolted to the floor and has conduit running to it. Mobile equipment such as pallets, bins, dies used for pressing, etc. are not defined in the production model, but configured in the MES Management screen as Supplemental Equipment (Track & Trace only).



Below are the different types of Production Items that can be added to the production model.

Icon	Production Item	Description	Module
	Enterprise	The enterprise is the highest level of the production model and typically represents a manufacturing company. You can rename the Enterprise production item to your companies name. You can only have one Enterprise item in the Production Model.	All
	Site	A site is a fixed geographical production location that is part of an enterprise. Separating your enterprise into multiple production sites allows for comparing OEE, downtime and production information between them.	All
	Area	An area is a physical or logical grouping of production lines.	All
	Line	A line is a collection of one or more cells and/or cell groups that work together to perform a sequence of process steps. Typically, the product flows from one cell or cell group to the next in sequence until the product, or sub assembly, being produced is complete.	All

Icon	Production Item	Description	Module
		Understanding how Operations schedules or controls a production run will help in determining whether cells should be grouped into a line or be considered lines themselves.	
	Location	A location item is the place where a sample is collected. This can be placed under an area or a line.	SPC
	Cell Group	A cell group contains two or more cells. Typically, these cells occur at the same time in the sequence of the line instead of one after another, causing the cell group to act as a single sub process or step within the production.	All
	Cell	The cell is a single machine, sub process or step required in the manufacture of a product. The product may be a hard product such as used in packaging, adding liquid or powder, etc. Packaging machines are a common example, but a cell applies to processes also.	All
	Storage Zone	A storage area such as a warehouse.	T&T
	Storage Unit	A storage unit located inside of a storage zone. For example, you may have a warehouse with bay 1 to 5.	T&T

Configuring the Production Model

The production model is configured within the Ignition designer and is accessed by selecting the **Production** node under Global in the project browser. From here your enterprise, site, area(s), line(s) and line cell(s), line cell group(s), storage zone(s) and storage unit(s) can be added, renamed and deleted.

It is extremely important to understand production OPC values have an OPC item path that matches the layout of the production model and that renaming production items can cause Ignition tags associated with a production item to stop being updated.

Adding a New Production Item

To add a new Production item, right-click on the Production model and select the **New Production Item** > **New Production xxxx** menu item.

Renaming a Production Item

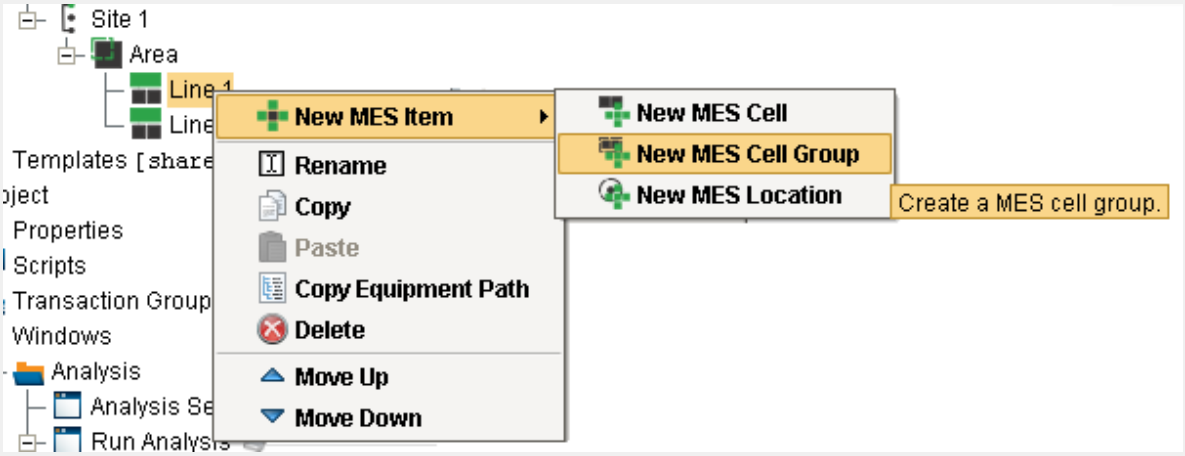
To rename a production item, right-click on it and select **Rename**, then enter the new name.

> Please note that when you rename a production item, it actually creates a new instance of a production item and disables the old production item. This is important to note as data captured against that production item will not be accessible to the newly renamed production item. Spend the time to get the Production Item named correctly at the beginning of the project.

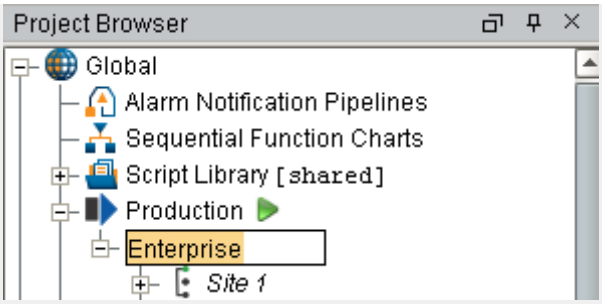
Deleting a Production Item

To remove an existing production item, right-click on the item and select the **Delete** menu item. A window will appear confirming that you permanently want to delete the production item.

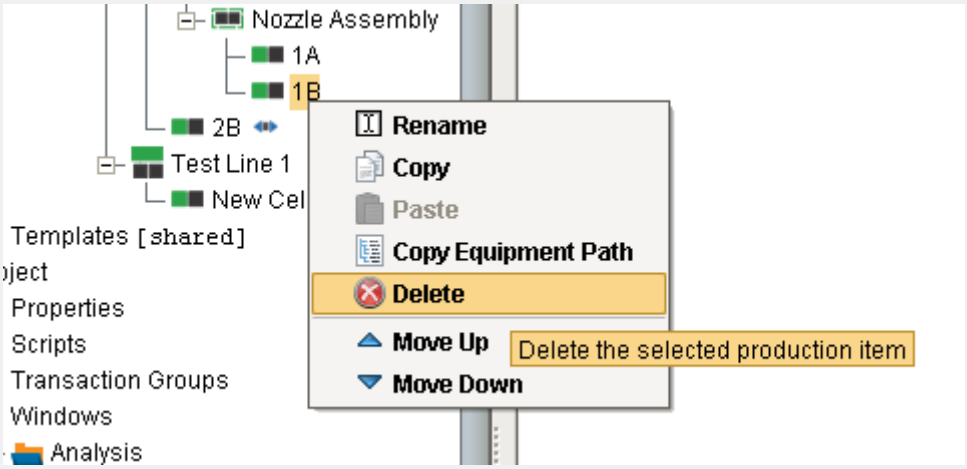
> Please note that any line(s), cell(s), cell group(s) and location(s) underneath the production item will also be permanently removed.



Adding a new Cell Group to the Production Model



Renaming the Enterprise



Delete a Cell

Copying a Production Item

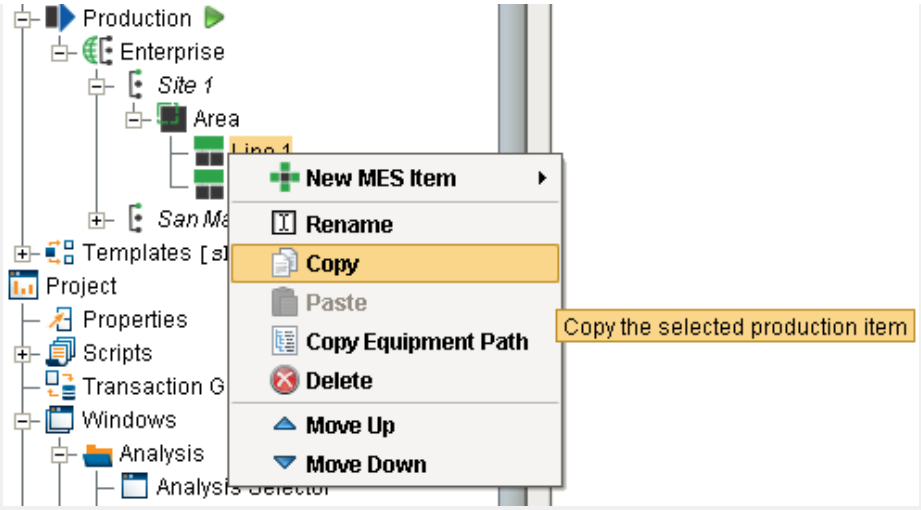
Right Click mouse button and select **Copy** on any production item to copy that production item.

Right Click mouse button and select **Paste** to make a copy of that production item in the production model.

If you are copying a line, select the line before copying it. When you paste it, select the area in which to the create a copy of that line.

Good Practice

It is recommended that you make a gateway backup prior to copying and pasting Production items. It is not recommended that you make changes to the production model on the production server without scheduling with Operations and having the system backed up.



Copying a Production Item

Production Item General Settings

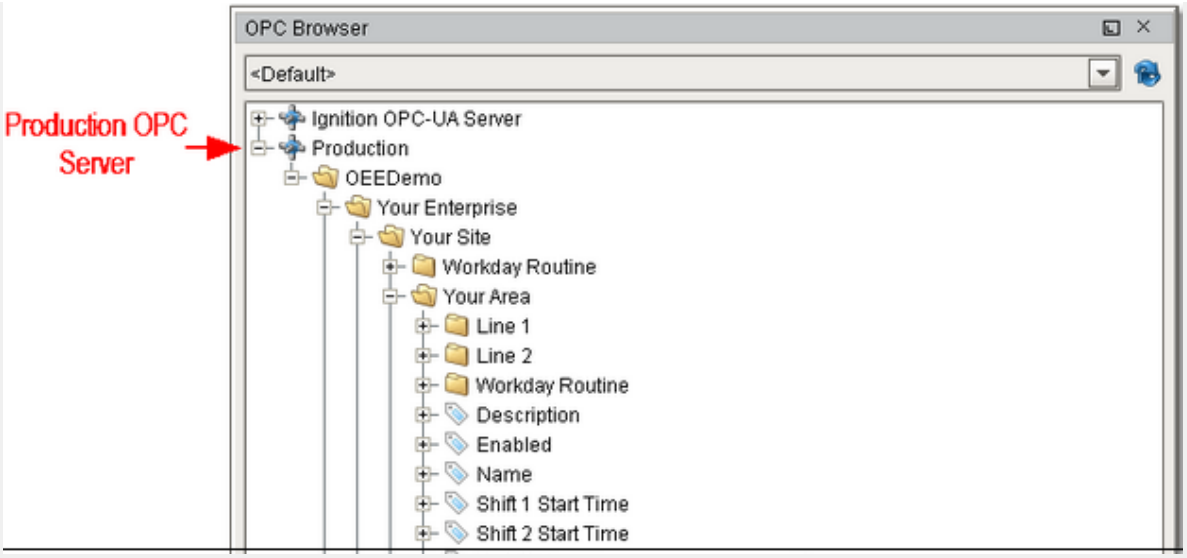
The general settings are accessed by selecting the desired production item and selecting the General tab.

Setting	Description
Enabled	By default, the production item is enabled. It can be disabled by un-checking the Enabled setting and saving the project. This will stop the track and trace, OEE, downtime, SPC, recipe and scheduling modules from using the area and any other production items that are underneath it.
Description	This is an optional description and is just for your reference.

OPC Production Tags

As production items are added to the production model, run time access into configuration settings and current state of those production items is available through the Production OPC Server. It is added automatically when MES Modules are installed. When the production items are added, removed or modified, the changes will be reflected in the Production OPC Server when the project is saved in the designer.

Please refer to the OPC Production Server Tag Reference in the Appendix for more help.

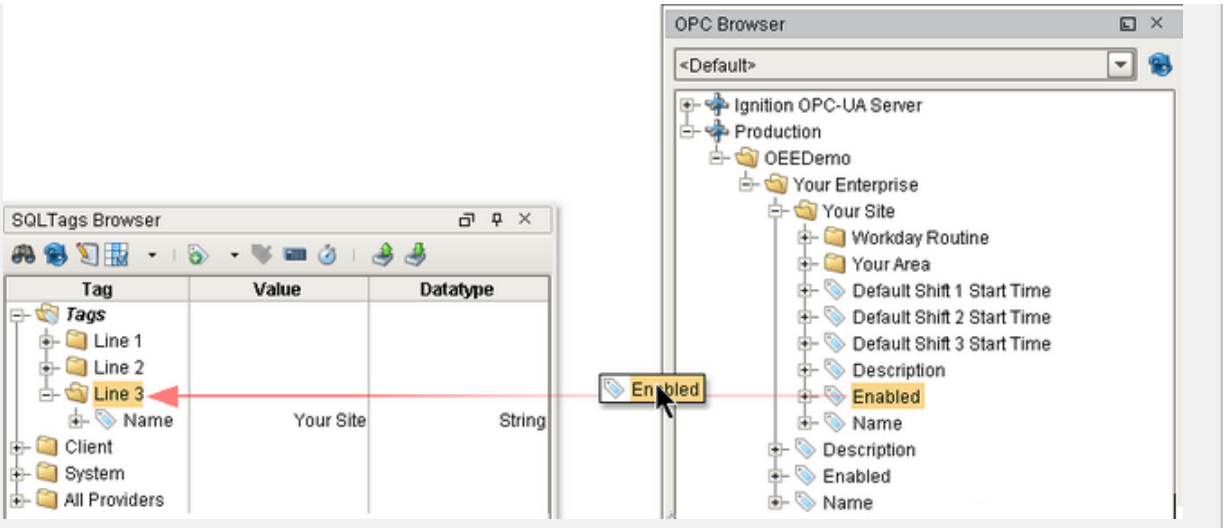


Demo OPC Values

Using OPC Production Tags in Your Project

Before Production OPC Server tags can be used in your project windows, transaction groups etc., they must be added to the Ignition SQLTags. This is done in the designer by selecting the SQLTag Browser and clicking on the OPC icon. This will cause the OPC Browser to appear. Next, drill down in the Production node within the OPC Browser. Drag the desired Production OPC Values over to the SQLTag Browser as shown.

When writing to OPC values that are related to production model settings, the new value is not retained upon restarting. This is because production model settings are saved in the Ignition project and is only saved when done so in the designer.



Add Production OPC Server Values to SQLTags

Setting Up Production Counters

In order to be able to calculate OEE Performance and OEE Quality, we need to provide at a minimum two counts, either outfeed and waste count, or infeed and outfeed counts at the line level. If all three counters are available for a line, then additional metrics can be generated such as actual transit time or accumulation (WIP). The following section provide specific details on adding Infeed, Outfeed and Waste Counters which are handled through the MES Counters in the Production Model Designer.

MES Counters

The MES Counters are used to associate Process Segments (Operations) with production counts. MES Counters record production counts 7/24, independent of scheduled production runs.

Counter names and the associated tag are defined in the Production Model. In the MES Management screen, the Quantity Source of Infeed and Material Process Segments can be set to use these MES counters.

MES counters are available for the Line, Cell Group, Cell or a Storage Unit production items in the Production Model Designer.

It is recommended to set up MES Counters at the cell level in order to accurately capture counts while indexing product on the line. Addition configuration must be accomplished with the cell settings in the production model.

The quantity from the MES counters can be obtained through scripting, see system.mes.getCountValue.

Adding MES Counter

To configure an MES Counter, select a Line, Cell Group, Cell or a Storage Unit in the production model and select the **General** tab on the right. Right click on the **MES Counter** table and select **New**. Properties can be set through the **Add MES Counter** Window.

Counter Name

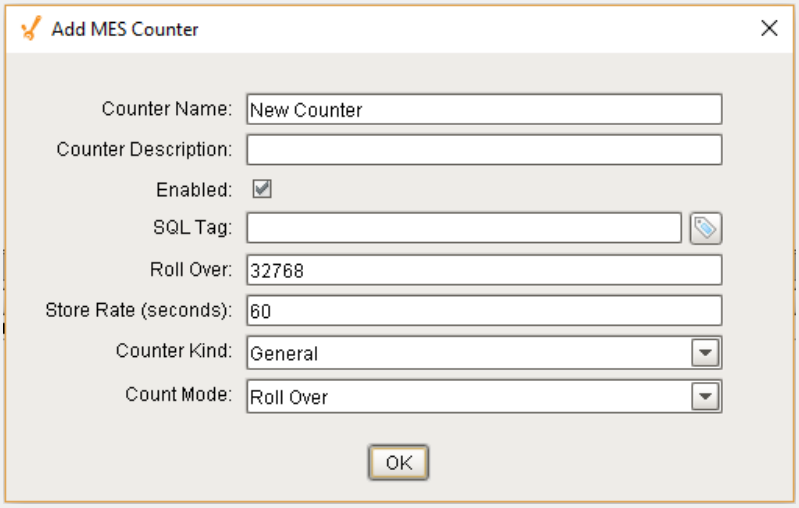
Name of the counter.

Counter Description

The description for the MES counter. This setting is not mandatory.

Enabled

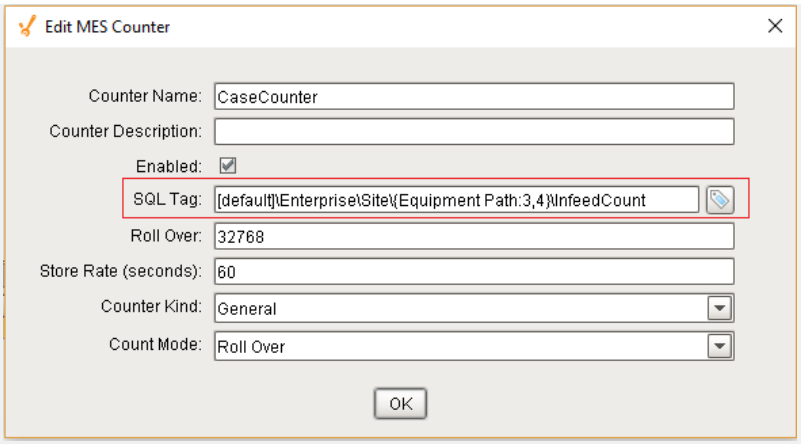
The counter can be enabled or disabled here.



SQL Tag

The path to the Tag Provider and ignition tag where the count value will come from is assigned to the MES counter here.

Parameterized Tag Paths can be used here which allows for indirection and exported MES Counters to be easily deployed to other equipment.



Roll Over

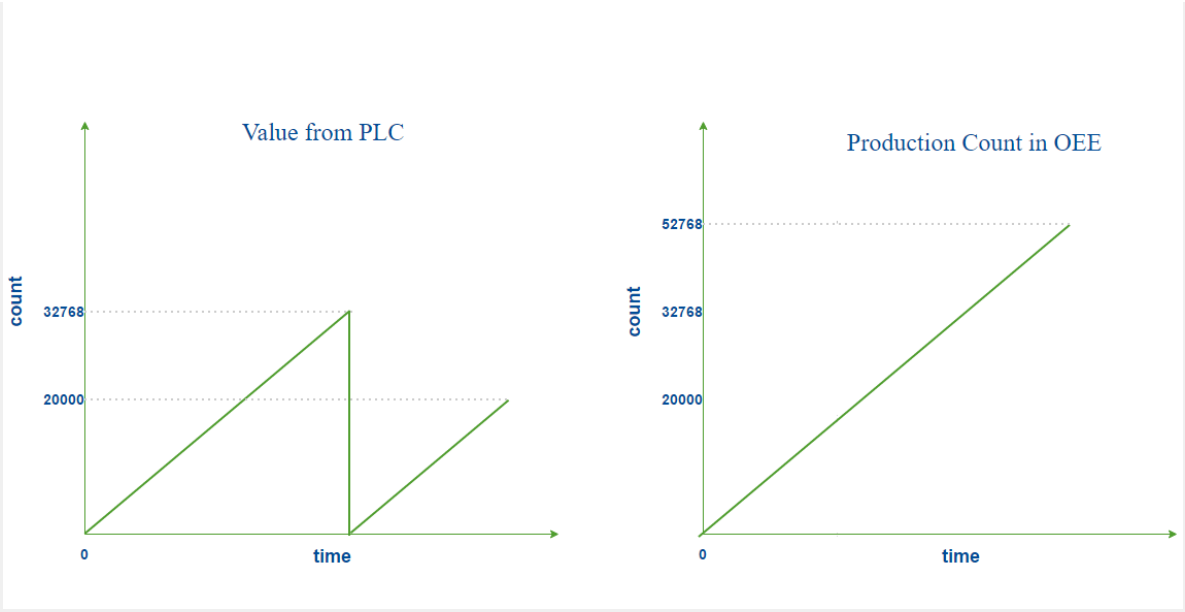
For PLC count tags that that do not get reset, they will eventually reach a finite maximum value at which point, the value will 'rollover' back to zero. The Roll Over setting allows you to define the value that should be added to the count tag whenever a roll over occurs. By default it is 32768 which equates to a 16 bit signed data value. Your setting will be dependent upon the datatype of your plc count tag.

During a production run, the incoming count value is added to the Roll Over setting multiplied by the number of times a rollover has occurred.

Example: production count value = incoming count value + 32768 * 3

Production Count	PLC Count tag	Rollover	Calculation
32700	32700	0	32700 + 32768 * 0
32750	32750	0	32700 + 32768 * 0
32918	150	1	150 + 32768 * 1
33068	300	1	300 + 32768 * 1

This value is only used when the **Count Mode** is set to **Rollover**.



Roll Over Count Mode

Store Rate

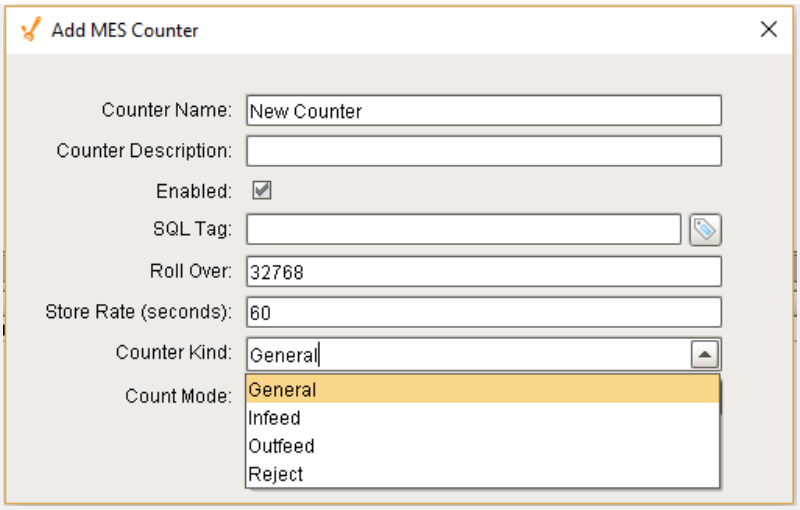
The MES counter will be captured and stored in the database after this specific interval in seconds if the value has changed If **Store Rate** is set to zero, every value change will be recorded.

Counter Kind

MES Counters can be set to four different kinds:

- Infeed
- Outfeed
- Reject
- General

Infeed, **Outfeed** and **Reject** kinds are used solely by the OEE module to determine which MES counter to use for OEE Performance, OEE Quality and production count information. The **General** kind can be used for any other count value.



Count Mode

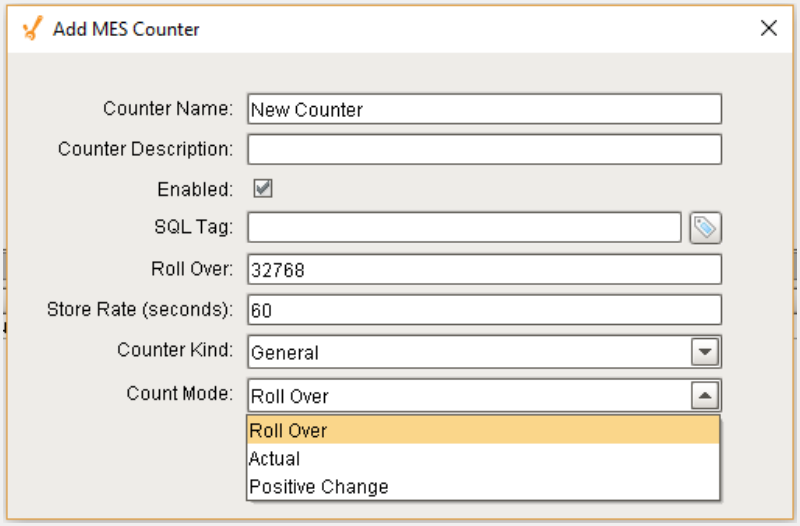
The Count Mode can be set to **Roll Over**, **Actual** and **Positive Change**.

Roll Over

See Rollover section for this count mode.

Actual

The Actual count mode simply uses whatever value is passed through the sql tag to represent the actual production counts. Production counts can go down as well as up.

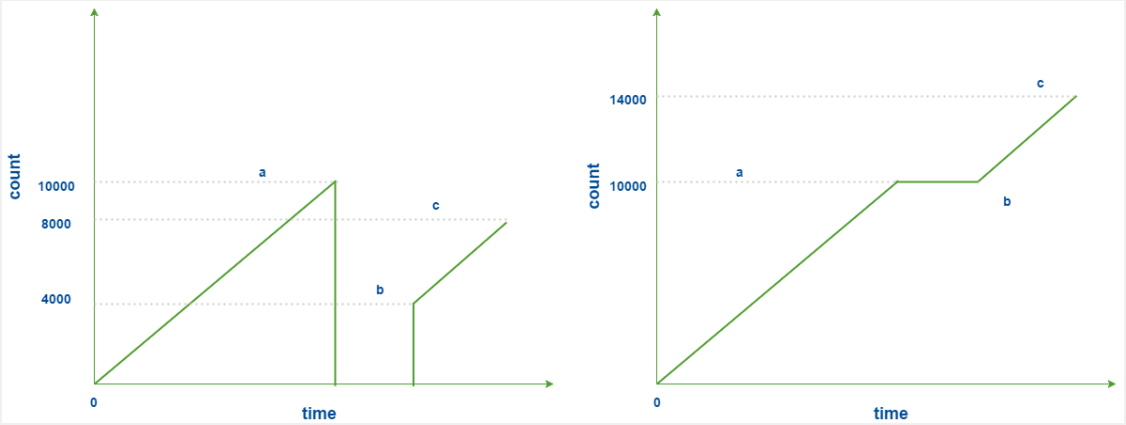


Positive Change

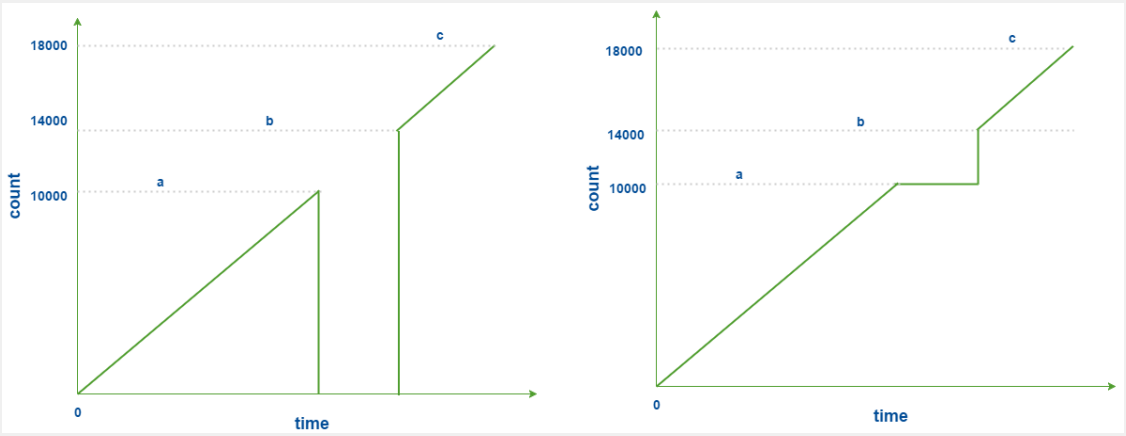
The Positive Change mode ignores any sql tag count values that are zero and will accumulate the counts. Three different cases are illustrated using the graphs shown.

PLC Count vs. OEE Count

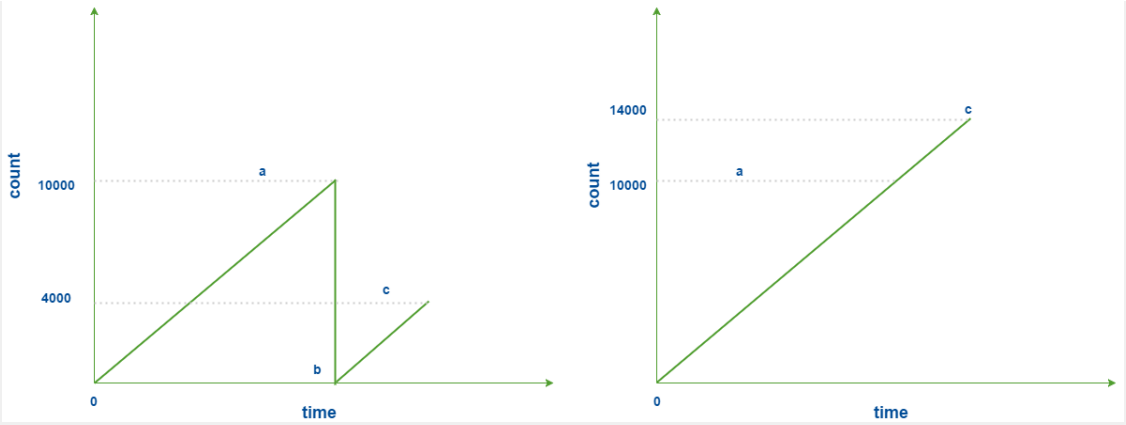
Case 1



Case 2



Case 3



Counter Rapid Development Features

Not only do counters allow for parameterization (as shown above), they also support copy, paste, import, and export features for rapid development. Configure the infeed counter for one Production Model Node (i.e. a Cell or Line) with parameterization, then copy and paste it to the other cells. Alternatively, copy and paste the Node itself and rename it for a similar effect.

Production Count Scenarios

There are a number of ways that production counts can be captured for a line. Understanding how to capture, what to capture and the ways that production counts can be captured is an important step in configuring the production model to ensure that we obtain accurate OEE metrics. The video discusses a number of scenario's regarding how production counts can be configured and tracked.

Sources Of Production Counts

There are different sources to read the production counts. With the OEE Downtime Module, production counts can come from the PLC, a database, manually entered, barcode reads and so on. The video discusses a number of common sources for production counts.

Best Practices for Production Counters

Tags are required from the lines devices that will provide the counts needed. These count tags are known as **Raw Counts**.

> The term **Raw Count** is used because it is a relative production count. It just starts at zero and counts up to a rollover value, typically 32767, where it becomes zero again. The OEE module calculates the actual production count from raw count. This eliminates having to reset the value in the PLC, or other device, at the beginning of a production run. As a result, the programming that is required in the PLC, or other device is simplified. It also eliminates problems typically associated with reset handshaking and production runs that exceed the limits of PLC counters. For an OEE tracking system to be accurate, it must withstand communication errors power outages, etc. By using raw counts that rollover and let the OEE module handle the actual production count, the system is robust. Besides that, it is just less PLC programming that has to be done and tested .

The OEE Engine calculates a **Relative Production Count** based on the beginning value of the **Raw Count** tag at the start of the run.

For example, if the Outfeed Count tag value is 100 at the start of the run, 100 becomes the baseline value.

Relative Production Count value will be calculated as ***Raw Count*** *current value* - ***Raw Count*** *baseline value* throughout the run.

Handling Rollovers

Max Raw Count

Each of the Counters, Infeed, Outfeed and Waste can have a **Max Raw Count** value defined to handle a rollover or overflow condition. Based on the datatype of the tag within the plc, the rollover will occur when the maximum value the register can handle has been exceeded and the next increment resets its value to 0.

DataType	Bits	Max Value
Signed Integer	31	2147483648
Unsigned Integer	32	4294967296

Depending on the age of the PLC and the register type used, the rollover count may be much lower. 2^{15} = 32,768.

The **Max Raw Count** setting handles the rollover by adding the value defined for the **Max Raw Count** to the value of the **Relative Production Count** whenever a rollover condition occurs.

Example

if **Max Raw Count** is set to 32,767 and the **Raw Outfeed Count** tag value changes from 32,765 to 5, the **Relative Production Count** recorded by the OEE Engine will be 5 + 32,767 (**Max Raw Count**).

This works the same if **Max Raw Count** is set to 0 or to 1... the **Relative Production Count** recorded by the OEE Engine will be ***Raw Outfeed Count*** *tag value* + ***Max Raw Count***.

Non-Resetable Counters

The counters coming from the PLC must be non-resetable from the HMI or by an operator as we cannot control when an operator might chose to reset a production or shift counter . If the value suddenly changes from say 100 to 0, the OEE engine will assume that a rollover has occurred and the production count recorded will be in-correct.

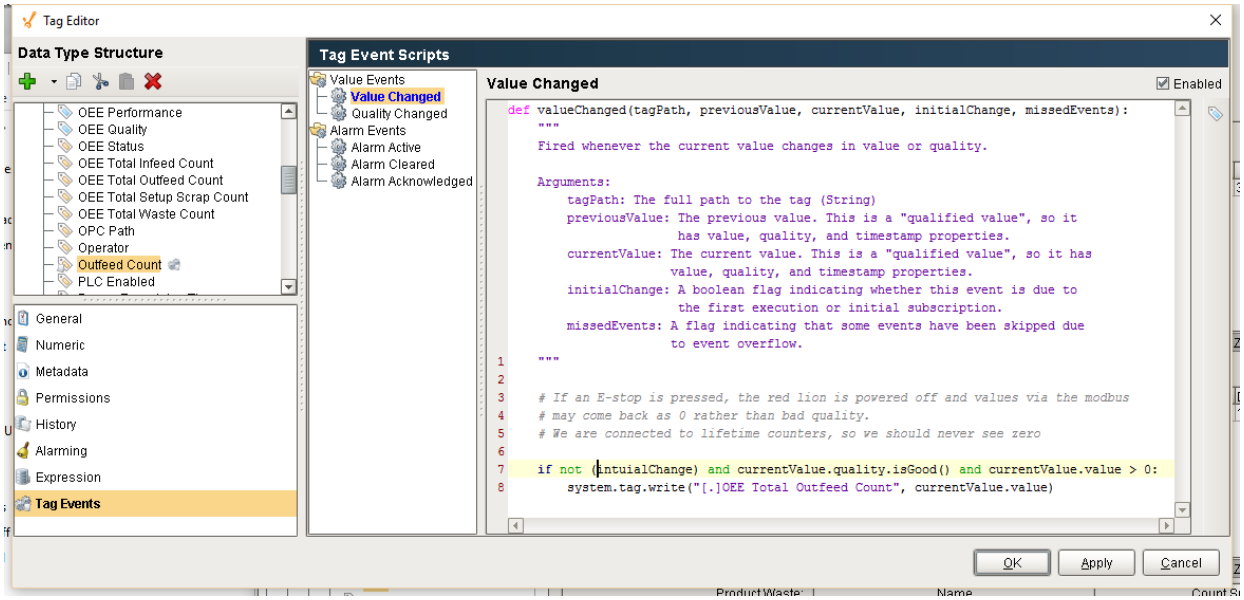
Ideally, you will have or create lifetime counters within the plc logic to manage production counts. The occurrence of a rollover should be an infrequent occurrence. In this case, you can set the Max Raw Count value to be the max value for the count tags coming from the plc when those counters tags are non-resetable.

If the lifetime counters are functioning correctly, if the plc logic has been written correctly so that a count tag does not yield an invalid value halfway through the scan, and if the device connection is solid, then the **Max Raw Count** will handle the rollover correctly. Otherwise, you may see large jumps throughout your production run and your production count will be wrong. In that case, having a Max Raw Count of 0 will allow for sloppy plc code and poor device connection, but it will not handle the rollover when it occasionally occurs.

Count Validation Techniques

The Production Model Count fields for Infeed, Outfeed and Waste can be bound directly to a count tag from a PLC. However, a good practice is to use an expression tag to bind a tag to the **Raw Count** coming from the PLC and to then use a Tag Change Event to validate the count before passing it to a memory tag associated to the production count in the Production Model. This provides the flexibility of deriving a calculated value if the count is variable (see section on

Variable Production Count below) as well as allowing for count validation prior to passing the value to the production model. In the code example, we are checking the tag quality to minimize issues caused by OPC connection issues. we are also checking to ensure the count value is not negative. The validation you provide is up to you. the Tag Change event makes it possible.




```
Tag Change Event Count Validation

if not (initialChange) and currentValue.quality.isGood() and
currentValue.value > 0:
    system.tag.write("[.]OEE Total Outfeed Count", currentValue.
value)
```

High Volume Manufacturing

If your line is a high volume manufacturing process, where rollovers may be a common occurrence, it is possible to implement counters that are only resetable by the OEE module.

In the Advanced tab of the Production Model at the Enterprise level, we can define scripts that occur on process segment '**Begin**' and '**End**' events. These events are generated by the production model. These events could be used to reset the production counters at the end of each run. That would most likely eliminate rollovers from ever occurring, in which case Max Raw Count could be left at 0.

Variable Production Count

You may have a Production count coming from a PLC that provides strokes from a Stamping Press but you want to count the # of pins produced and the # of pins is dependent upon the Die being used in the Stamping Press. In this case, the **Max Raw Count** at the Production Count level will not help you as it becomes variable.

The Infeed Scale parameter in the Material Manager and the Infeed Scale Tag path allows you to set the scaling factor for the count.

Adding Additional Factors

The OEE Module collects and logs a number of downtime and production data values. However, what if other values outside of downtime and production values are of interest? Additional factors are the solution. Additional Factors are user defined data points that are logged along with the production and downtime information. Once they are logged, they can be shown in charts, tables and reports. Additionally, other analysis can be done by filtering and/or setting up comparisons by their values.

Additional Factors can added to the Line, Cell Group and Cell Production Items in the Production Model Designer.

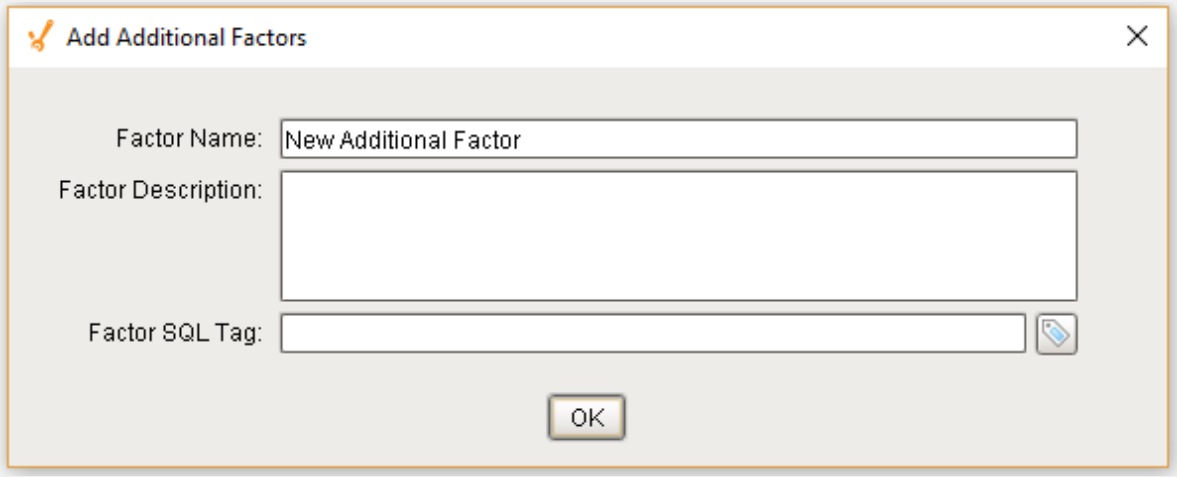
Any value that can be read from an Ignition SQLTag can be added as an additional factor. This includes values derived from scripts, or from barcode readers, databases, calculations, PLCs, etc.

Any tag can be added as an additional factor. To configure, select a Line in the production model and select the **General** tab on the right. Right click on the **Additional Factors** table and select **New**. An additional factor is simply just a name and a tag.

Properties

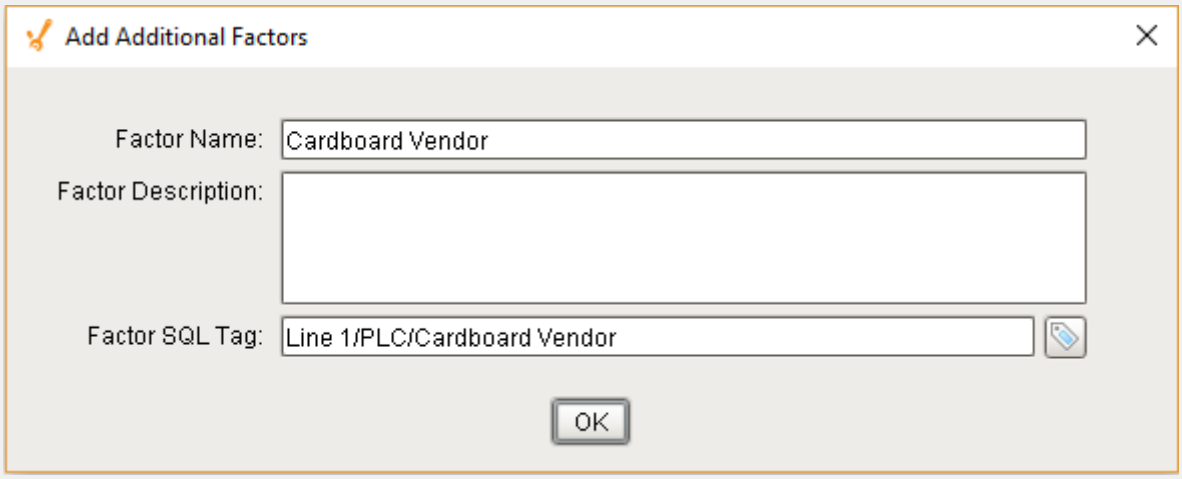
Factor Description	Optionally, this property can be set to a description for the additional factor. It is not used by the OEE Downtime and Scheduling Module other than for reference.	String
Factor Name	This reflects the name of additional factor that is configured in the designer.	String Read Only
		String

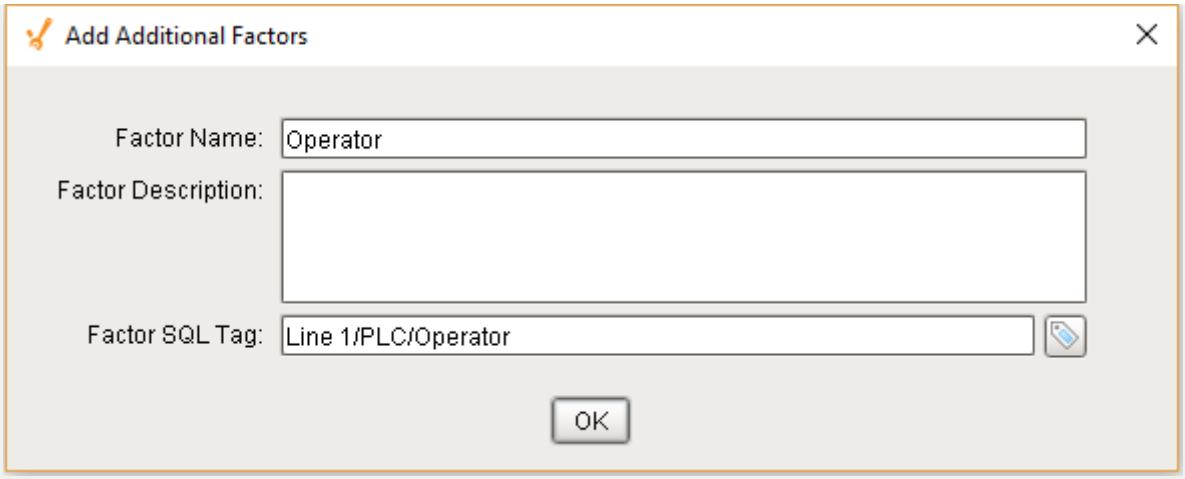
Factor SQLTag	This reflects the Factor SQLTag setting that additional factor is configured for in the designer. It is the name of SQLTag to read the factor value from.	Read Only



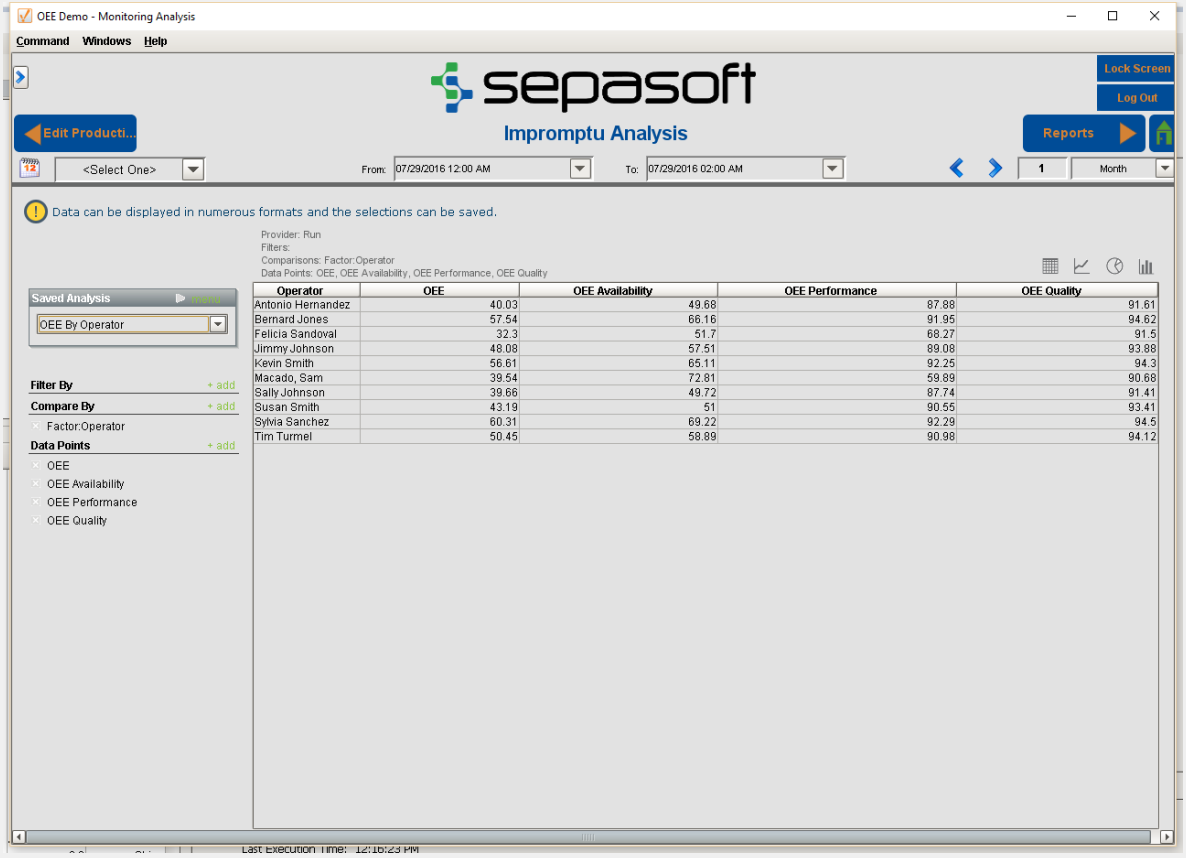
Example

In the example, we have two factors, **Cardboard Vendor** and **Operator**. The operator can select the vendor that provided the cardboard or it can be obtained from some other source. Now, OEE and downtime results can be shown for each cardboard manufacturer. This can identify quality problems with raw material that directly affect production efficiency. With the operator setup as an additional factor, the operator's name will be logged along with the production and downtime data. By doing so, OEE and downtime information can be filtered and grouped by the operator name. But this could just as well be the production crew, supervisor, maintenance crew or any other user defined value that can be monitored or entered into the system.





Adding these factors to a production line will allow us to capture the value of these tags whenever they change. In the impromptu analysis, we can then compare OEE values by our additional factor: Operator



Setting Up Equipment Modes

In manufacturing, it is fairly common for a production line to be running, but not actually producing parts or finished goods as part of a work order or product code run. Examples of this may be when the line is being run during scheduled maintenance to verify work has been performed to a satisfactory level. There are times when a line may be running as part of new product introduction or testing, or times when operator training is occurring. When the line is getting ready for a production run, there may be a period of time that the line is in a changeover or setup mode, and there may be times during a production run, when the line mode changes back to setup say after a fault, and material has to be run back through the extruder cell.

In OEE 2.0, it is now possible to define modes that the line or equipment may be in and you can define whether production counts should be included during these modes and used in OEE calculations. Modes allows for a more granular analysis of equipment utilization; how much time does a line spends in maintenance, training, changeover, setup, while still allowing the equipment states of running, blocked, idle, faulted to be captured. As production counts and equipment cycles are captured in all modes, you can still use these counters to determine run hours on equipment, strokes on die sets etc., that may drive maintenance scheduling and other activities.

Default Equipment Modes are provided for IDLE, CHANGEOVER, PRODUCTION, MAINTENANCE, TRAINING, TESTING. Custom modes can also be added.

The OEE Equipment Manager component is used to change equipment mode class, state and schedule.

The mode the equipment is automatically derived based on the settings as defined in the OEE Material Manager when a production run is started through the OEE Run Director or MES Schedule View component. The current mode can also be overridden by passing a tag value through the mode tag collector. See OEE 2.0 Downtime Tab - Equipment Mode for more details.

Equipment Mode Class

It is possible to create Equipment Mode classes that contains a set of Equipment Modes. The Equipment Mode class contains a number of pre-defined modes that you can use for all your Lines or equipment. By default, we provide you with the **Default Equipment Mode Class** that contains the following Equipment Modes....

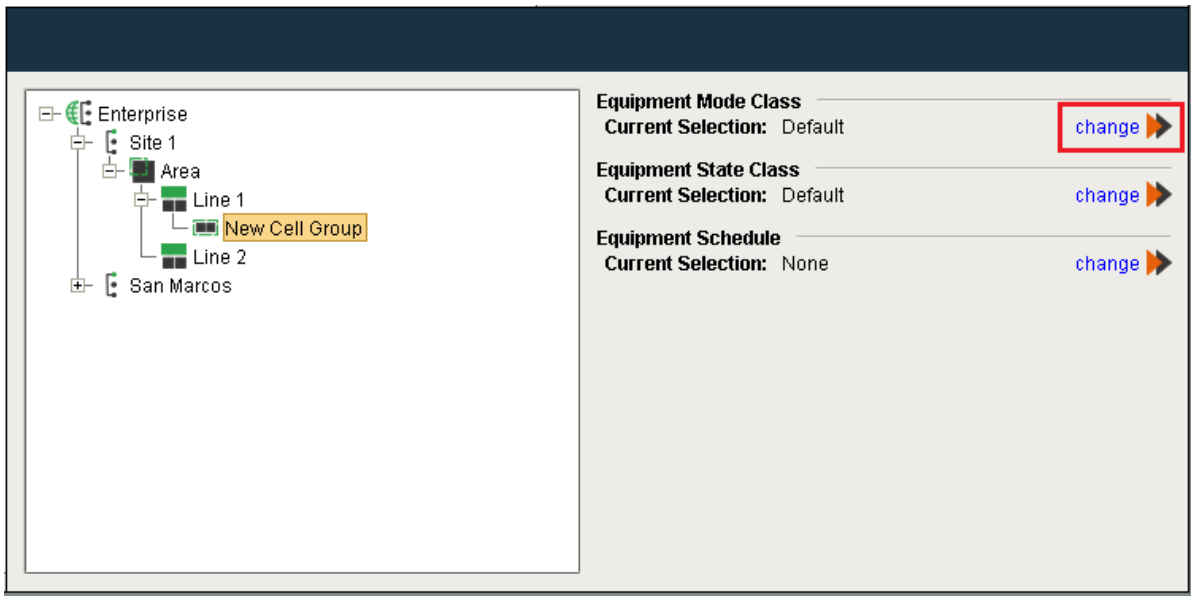
- IDLE
- CHANGEOVER
- PRODUCTION
- MAINTENANCE

- TRAINING
- TESTING

You can add or modify the equipment modes in the default class or create your own equipment mode class.

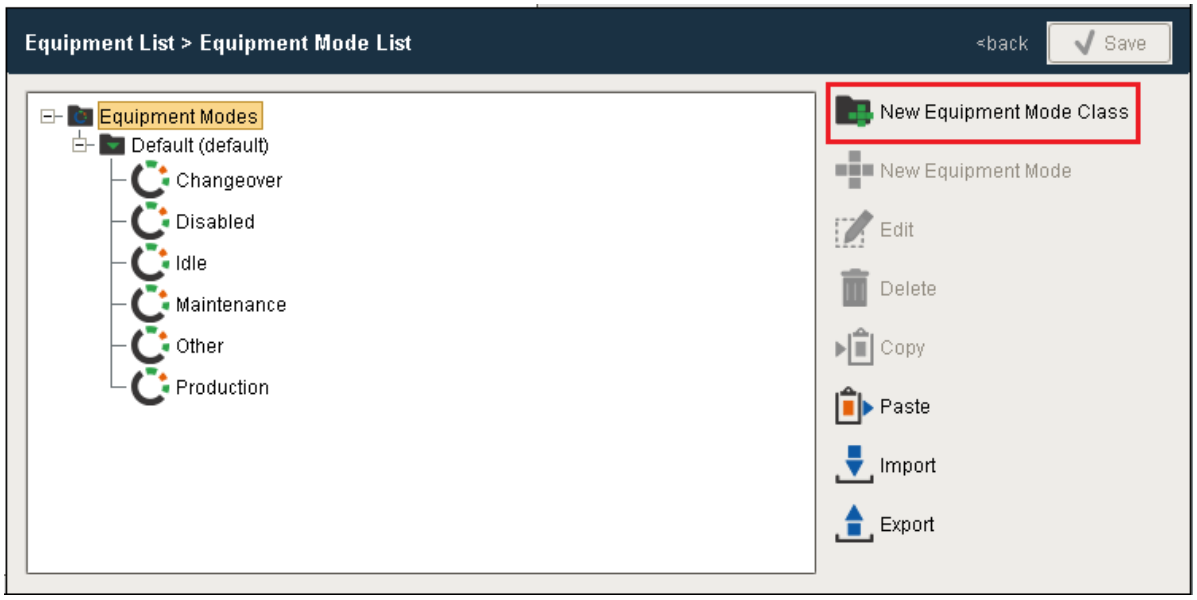
Adding and Editing an Equipment Mode Class

- Select the production model item for which the equipment mode class is to be changed.
- Click on the change button next to the equipment mode class.



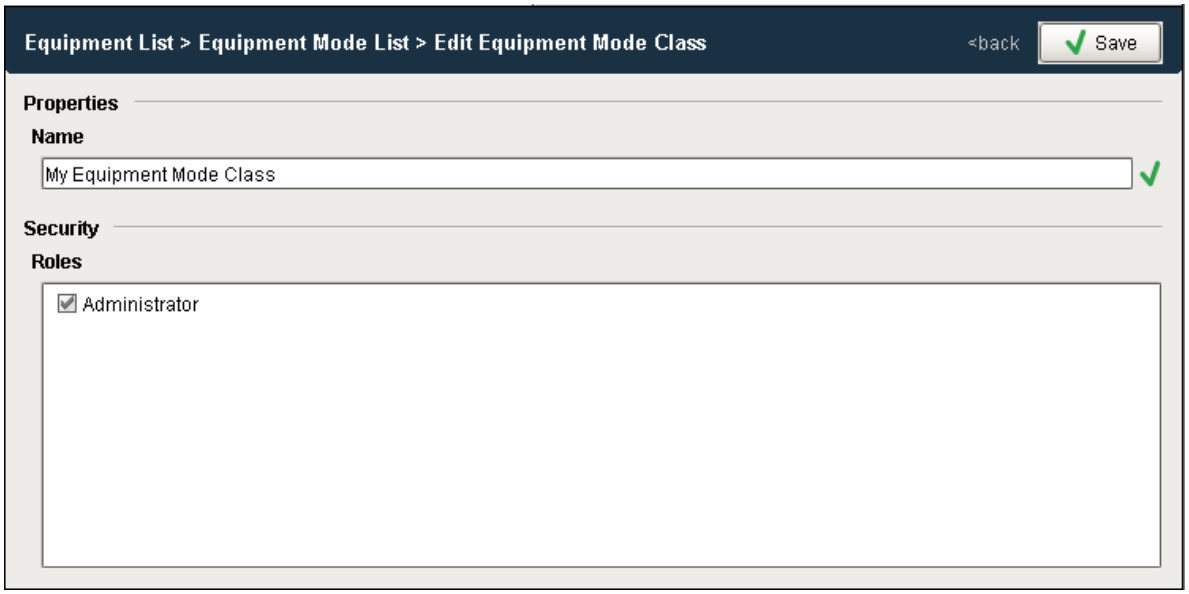
- The **Equipment Mode List** window appears. Click on Equipment Modes and select the **New Equipment Mode Class**.

> At the Equipment Mode Class level, **Edit**, **Delete**, **Copy**, **Paste** and **Export** operations are available. At the Equipment Mode level, only **Paste**, **Import** and **Export** operations are available.



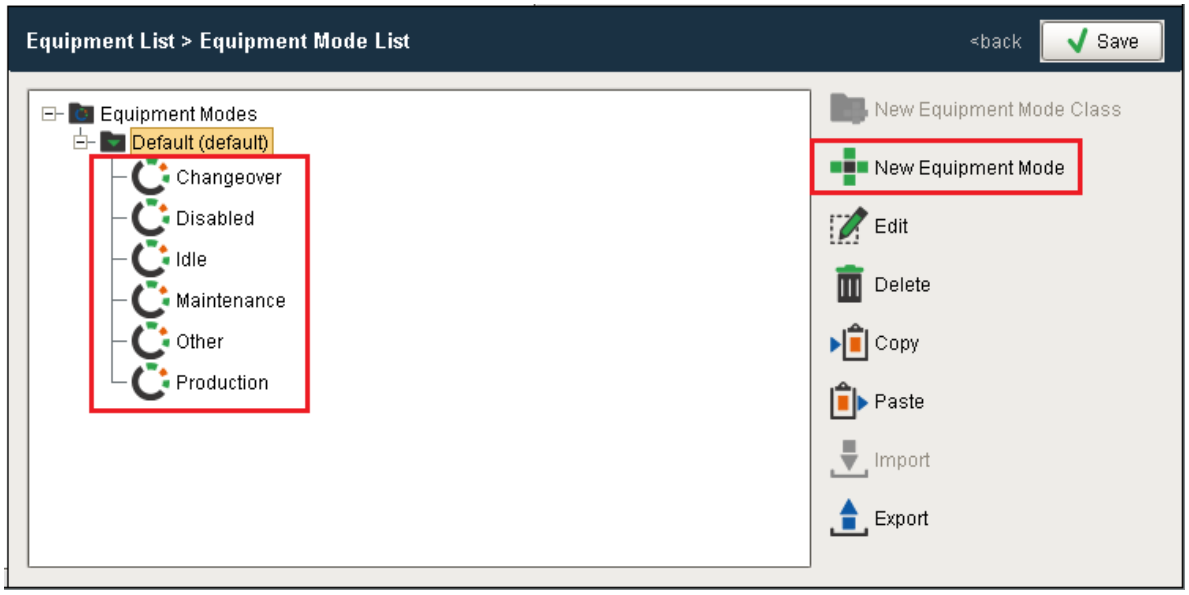
- Enter a name for the Equipment Mode Class and assign security roles you who can modify this Equipment Mode Class.
- Click **Save**.

We cannot add an equipment mode class to an existing equipment mode class, but we can add an equipment state class to an existing equipment state class.



Adding and Editing Equipment Modes for an Equipment Mode Class

- Select the Equipment Mode Class you wish to modify and click on the **New Equipment Mode** to add a new Mode.
- Select the Equipment Mode you wish to modify and click on the **Edit** to change the Mode.



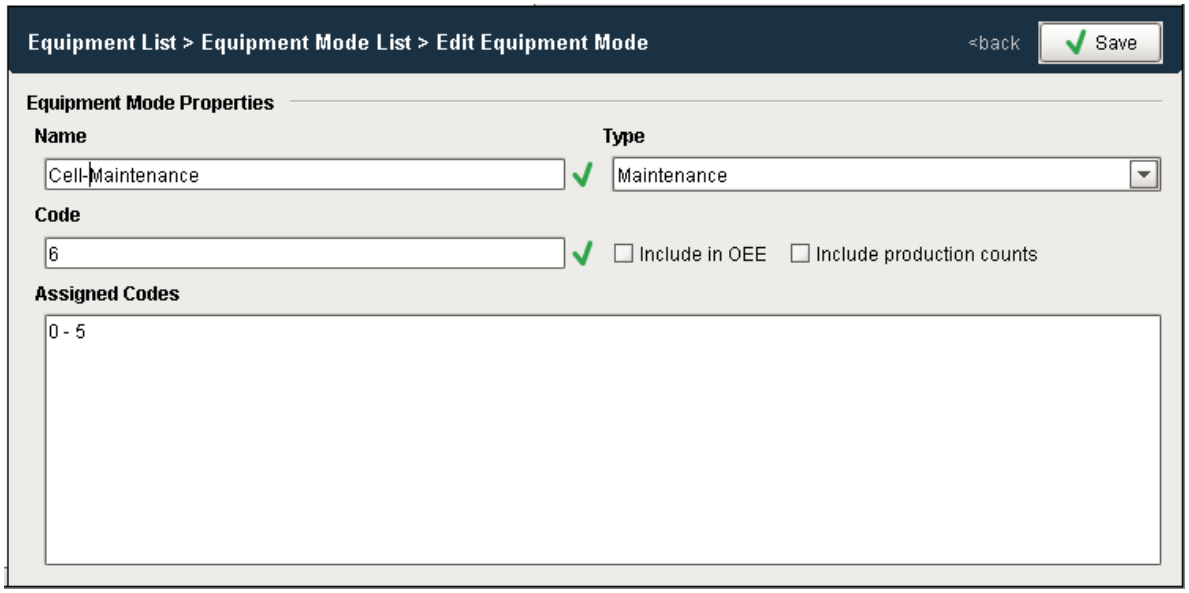
- The Edit Equipment mode slide out will appear.
- Enter a name for the mode and select the type of mode. Valid mode types are...
  - Production
  - Idle
  - Changeover
  - Maintenance
  - Other
  - Disabled
- Associate a code with the Maintenance Mode. This is used if you drive the mode of the equipment through a tag.
- Select whether production counts during this mode are collected
- Select whether counts and status during this mode are included in OEE calculations
- Click **Save**

Code

Enter in the integer value for this equipment state. This will generally align with the status code value from the plc.

Assigned Codes

The Assigned Codes list provide a view of all codes that have already been assigned



Include in OEE / Include Production Counts

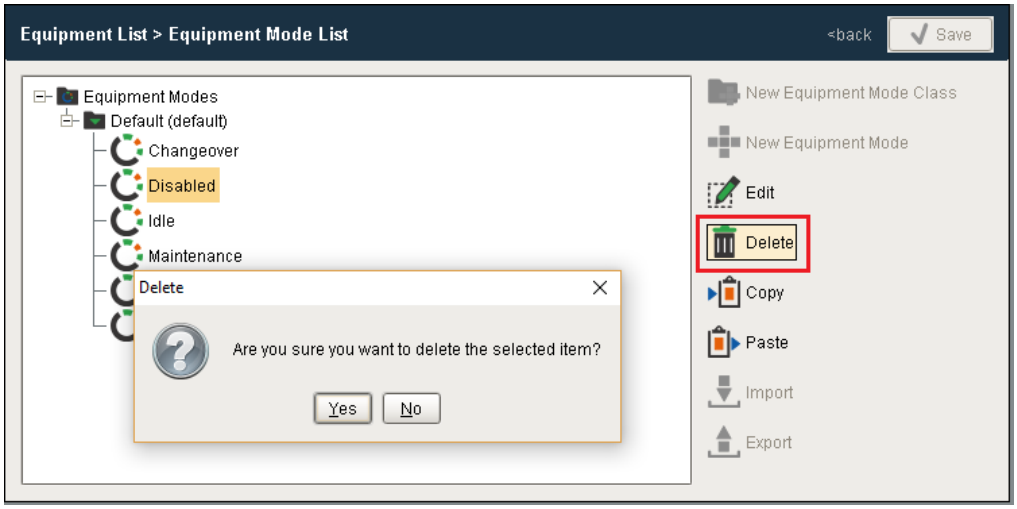
Two check boxes are provided that modify how Modes are considered by the OEE Engine. The following table provides valid settings for these two check boxes

Include In OEE	Include Production Counts	Description
		Use for modes such as cleaning or testing, where OEE and production counts are irrelevant
	✓	Use for modes such as Changeover or Setup, where production counts are still required but downtime etc. should not be included in OEE calculations for the line
✓		This is not a valid selection as production counts are required for OEE Performance and Quality Metrics

Include In OEE	Include Production Counts	Description
✓	✓	Use for mode such as Production where production counts and OEE data is required

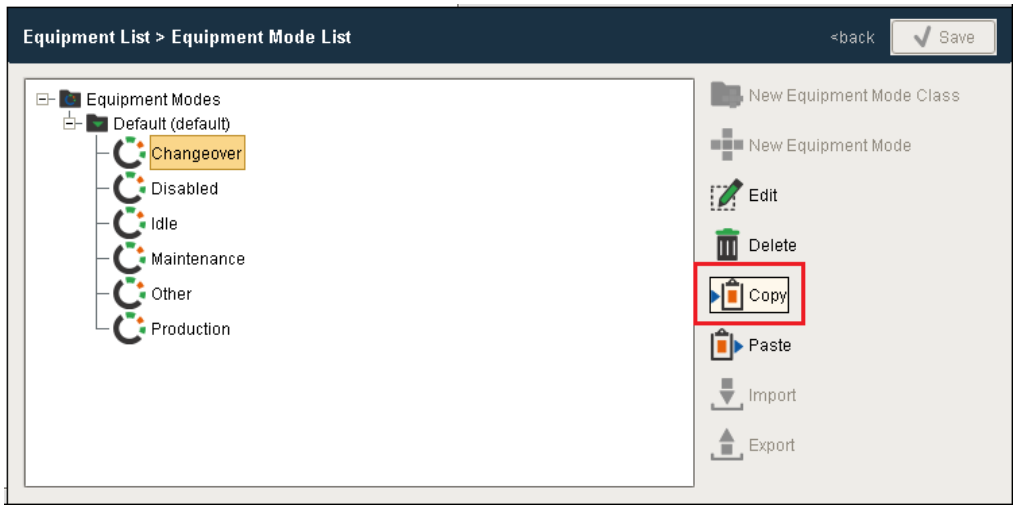
Delete Equipment Mode

- Click on the Equipment Mode to be deleted and select **Delete**.

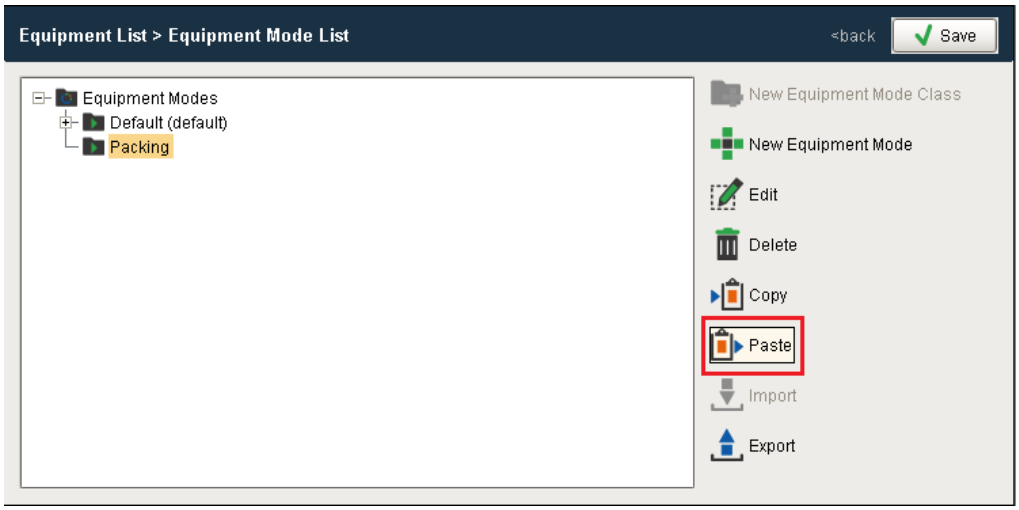


Copy and Paste Equipment Mode

- Select the **Equipment Mode** to be copied and click **Copy**.

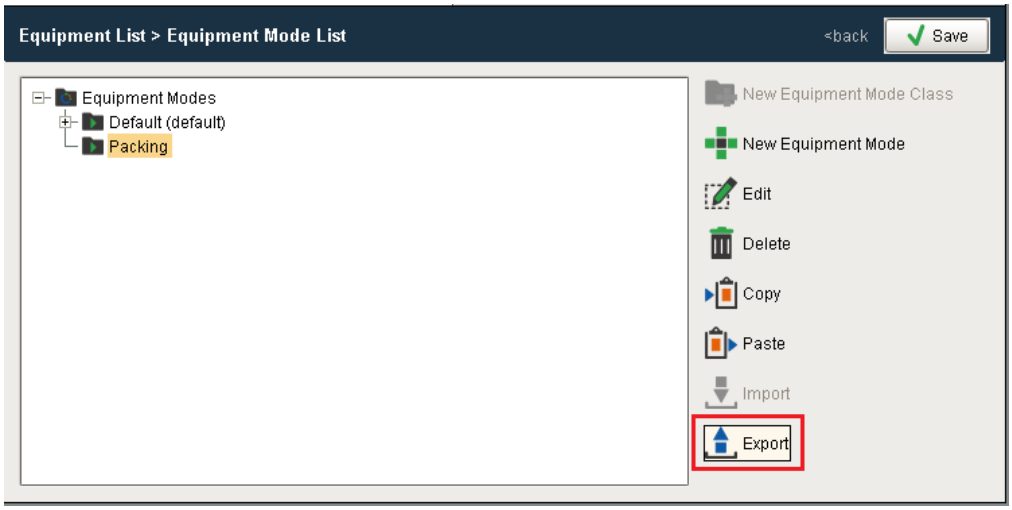


- Select the **Equipment Mode Class** to which the mode is copied and click **Paste**.
- Expand the destination folder to see the pasted equipment mode.



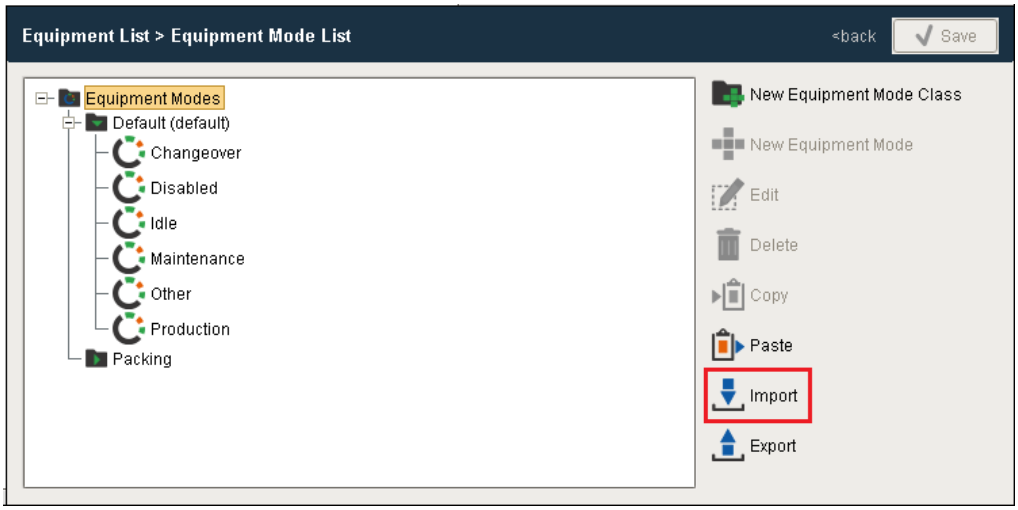
Export Equipment Modes

- Select Equipment Modes and Click **Export**.
- A file save dialgo box will appear allowing you to select where to store the xml file.



Import Equipment Modes

- Select **Equipment Modes** and Click **Import**.
- Select the file to be imported and Click **Open**.



Setting Up Equipment States

Equipment States represent the status of the line or cells within a line that provide an indication of whether the equipment is off-line, idle, running, faulted, blocked, starved, in CIP etc. It is entirely separate from the equipment mode, which provides a more logistics/scheduling view of equipment. The equipment status will generally come from a plc tag that provides real-time state information of the equipment, but can really come from any source that can populate an Ignition tag. This allows for manual entry screens of operator input equipment state or data parsed from a flat file entry.

Equipment States (formerly known as Downtime events) can now be grouped, allowing multiple states to be considered as **Running** by OEE i.e. Loading, heating, molding Running. No need to force your equipment state to 1 or group a bunch of states into one value. True equipment state is now captured and stored allowing for greater cycle time analysis.

Equipment states allows for the tracking of specific events that may prevent a line or cell from running. Some reasons are considered causes of line downtime where others are not. As an example, if a cell on a production line becomes blocked as the outfeed from it backs up and there is no room to discharge product it will stop. In this example, it is simply normal operation for the cell and not the cause of the production line not producing product. A cell further down the line is the cell preventing the production line from producing product. Other downtime reasons may be planned. Any time that the production line is scheduled around breaks, lunches, safety meetings, disabled shifts, etc., is planned and will not count against the production line OEE Availability.

The OEE Equipment Manager component is used to change equipment mode class, state and schedule.

The current state is captured from equipment via tag values passed to the state tag collector in the production model. Refer to OEE 2.0 Downtime Tab - Equipment State for more details.

Line Downtime Versus Cell Downtime

It is important to understand the difference between line downtime and cell downtime. Line downtime, which are downtime reasons that prevent the production line from producing product, is typically used to focus on improving OEE. Cell downtime is used to look at trends and detect maintenance issues before they cause line downtime. Consider a production line that has 25 cells. If 5 of the cells are down all at the same time for unrelated reasons and only one of them is preventing product from being produced on the line there will be a lot of noise (extra irrelevant data) to weed through. Also, if a faster downstream cell stops, restarts and catches up, it may never affect the production of the line as a whole. The OEE Downtime Module provides the best of both worlds and tracks both line downtime and cell downtime.

After an automatic reason has been triggered, the operator can override it with a more specific reason. Both are logged and can be viewed in analysis and reporting. For details about how to disable manual override see the **Editable** property in the Down Time Table section.

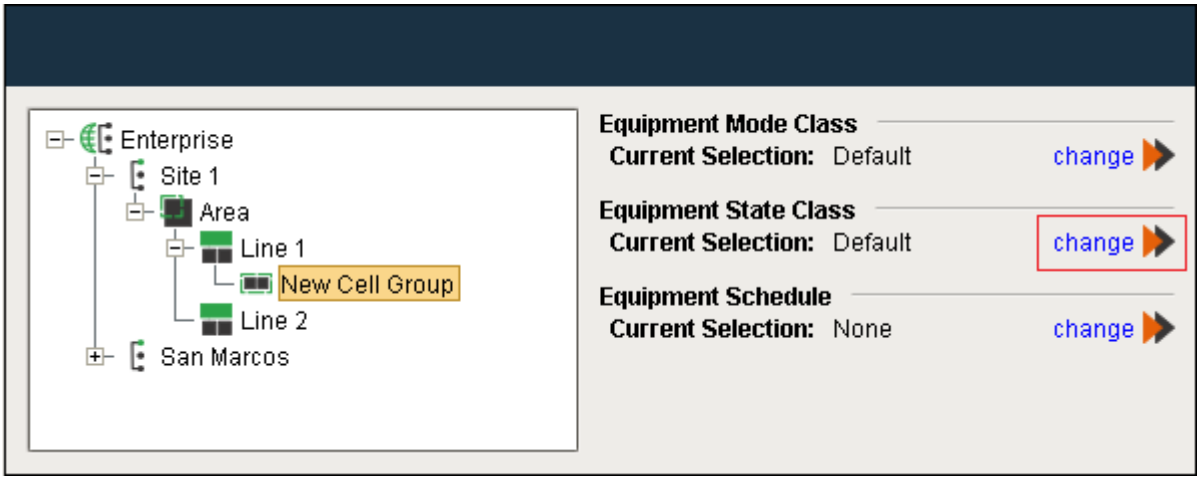
Obtaining Status Values for Cells on a Production Line

The OEE module determines the equipment state from a single numeric integer value. Single numeric values are stable and can only represent one state. There is no limit to the number of equipment states that can be defined other than by the maximum numeric value your PLC can handle.

When the OEE module detects a production line or cell state change, it will lookup the downtime reason from the state value. If communication to the PLC fails, in the case when an electrical connection is shut off, the production line or cell state is replaced with 0. If this happens during a production run, it will count as downtime if configured as such.

Adding and Editing Equipment State Classes

- Select the production model item for which the equipment state class is to be changed.
- Click on the **change** button next to the equipment state class.

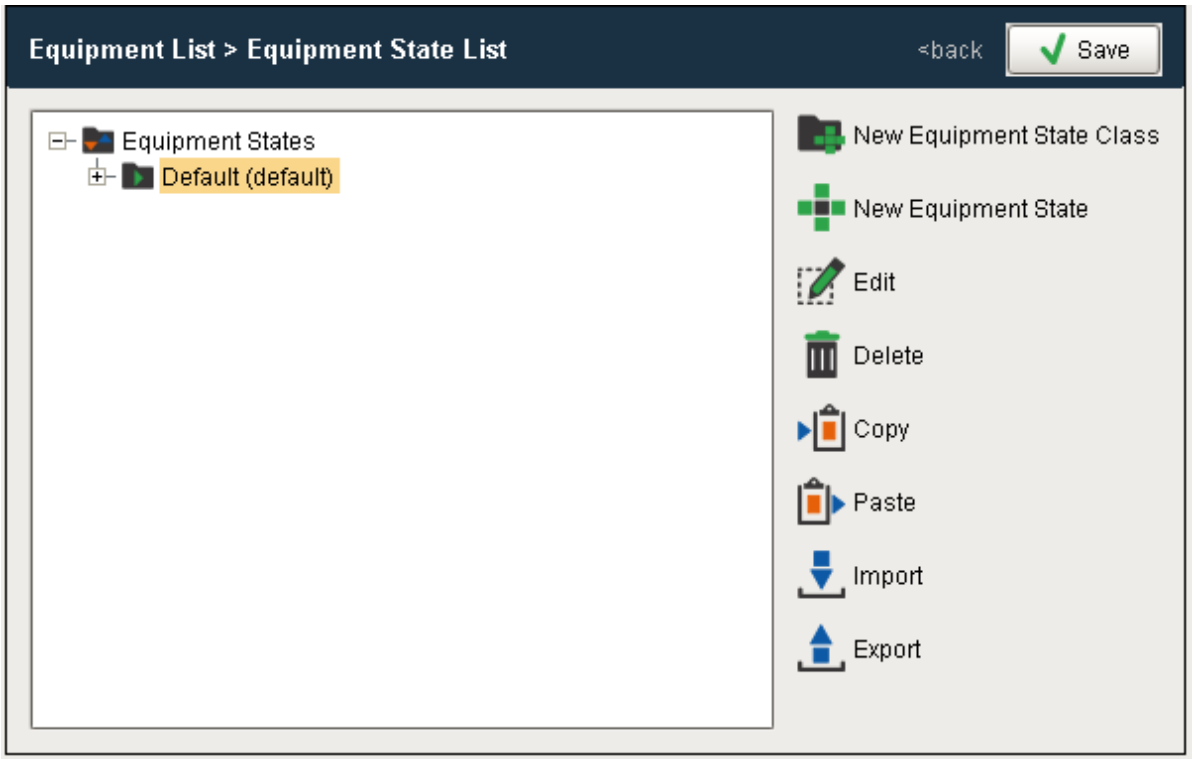


- The **Equipment State List** window appears.

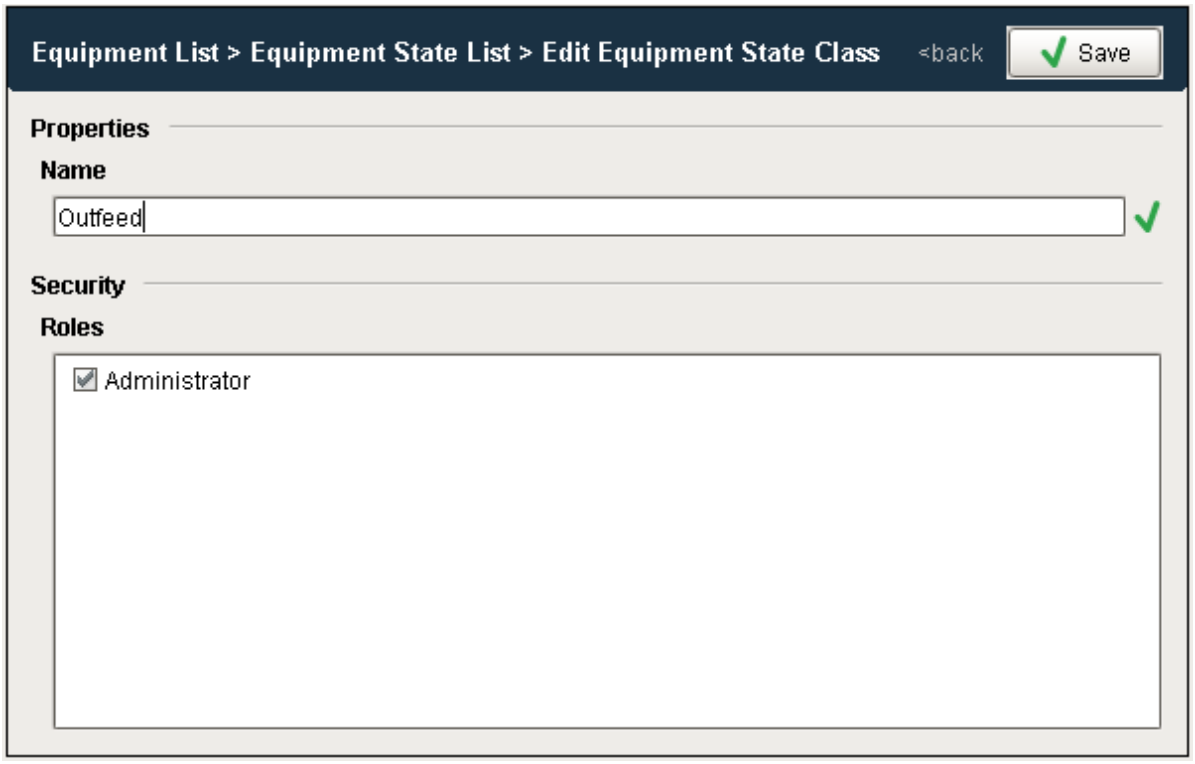
We can add an equipment state class to an existing equipment state class, creating nested state classes.

At the Equipment State Class level, **Edit**, **Delete**, **Copy**, **Paste** and **Export** operations are available. At the Root level, only **Paste**, **Import** and **Export** operations are available.

- Click on either an existing Equipment State or the **Equipment States** and then select the **New Equipment State Class**.



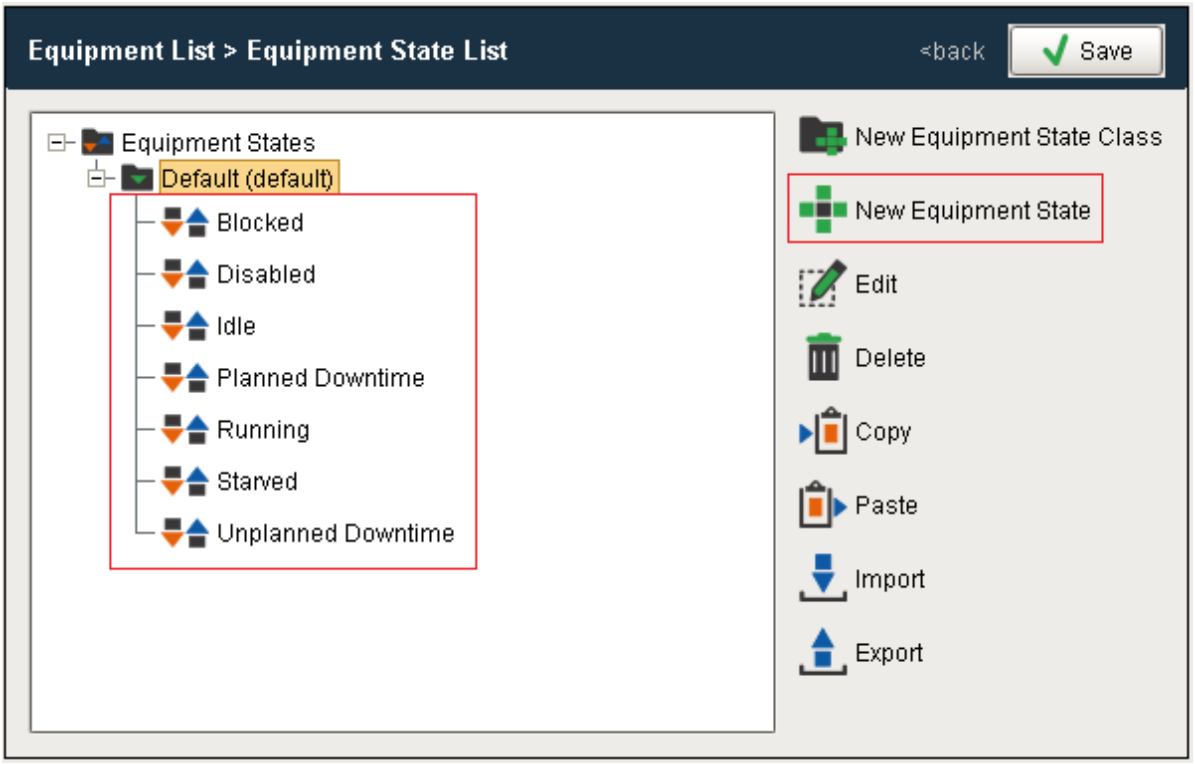
- Provide a name for the new State Class and assign security roles for who can modify this State Class.



Adding and Editing Equipment States for an Equipment State Class

- Adding - Select the Equipment State Class you wish to modify and click on the **New Equipment State** to add a new state.
- Editing - Select the Equipment State you wish to modify and click on the **Edit** to change the State.

If you expand the Equipment State Class folder you can see the existing states.

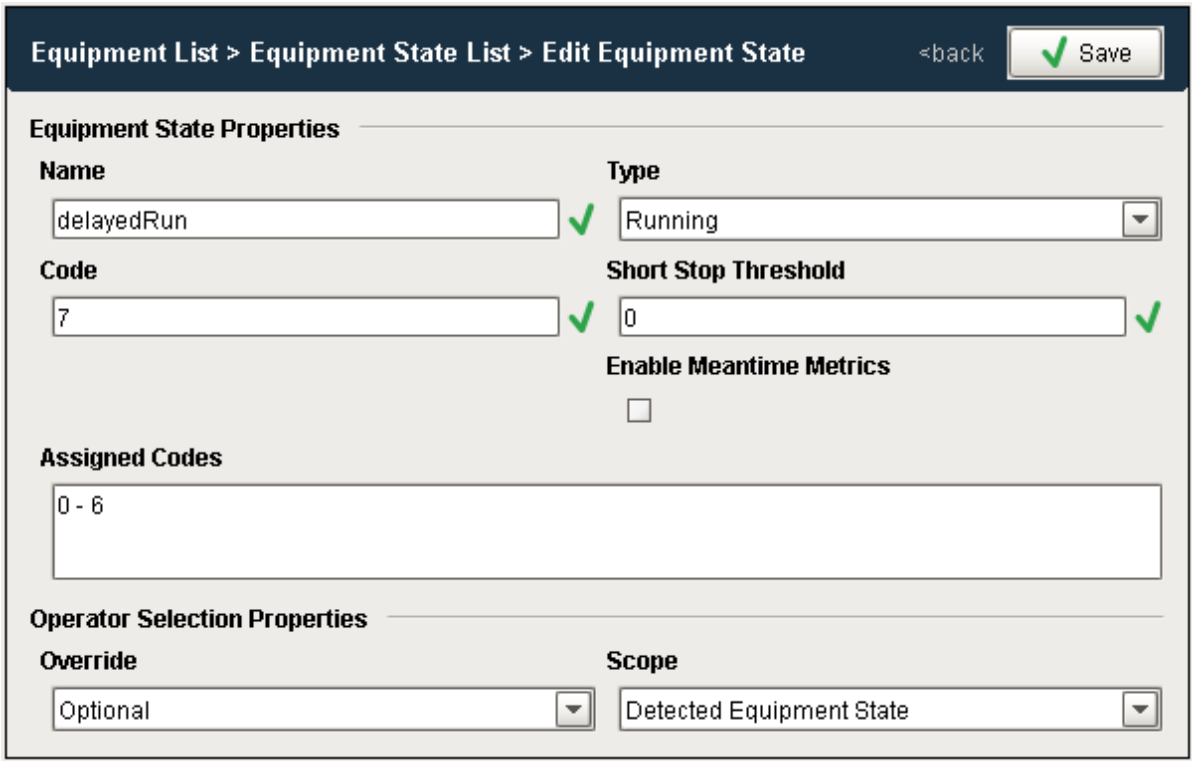


- The **Edit Equipment State slide out editor** allows you to change the properties of the selected Equipment State.
- Add or change the **Name**, **Type**, **Code,** and **Operator Selection Properties** and click **Save**.

State Type

Valid State Types are...

- Unplanned Downtime
- Planned Downtime
- Blocked
- Starved
- Running
- Idle
- Disabled



Code

Enter in the integer value for this equipment state. This will generally align with the status code value from the plc.

Assigned Codes

The Assigned Codes list provide a view of all codes that have already been assigned

Short Stop Threshold

Enter in a time value in seconds for how long a state must persist for before it considered to be in this. The Short Stop Threshold is not used for states of state type RUNNING.

Enable Meantime Metrics

When enabled, this state will be included in the calculation of Mean Time Between Failure (MTBF), when this state is responsible for causing a line downtime event.

Override

The Override property defines if it is possible for a operator to override the equipment state through the Downtime Table Editor component. Valid options are...

- Optional - Operator can select a different state based on the Scope setting
- Prohibited - Operator cannot change this equipment state
- Required - Operator is required to select a state based on the Scope setting for this equipment state

Scope

The scope setting is used by the Override setting to provide a set of operator selectable equipment states. Valid Options are....

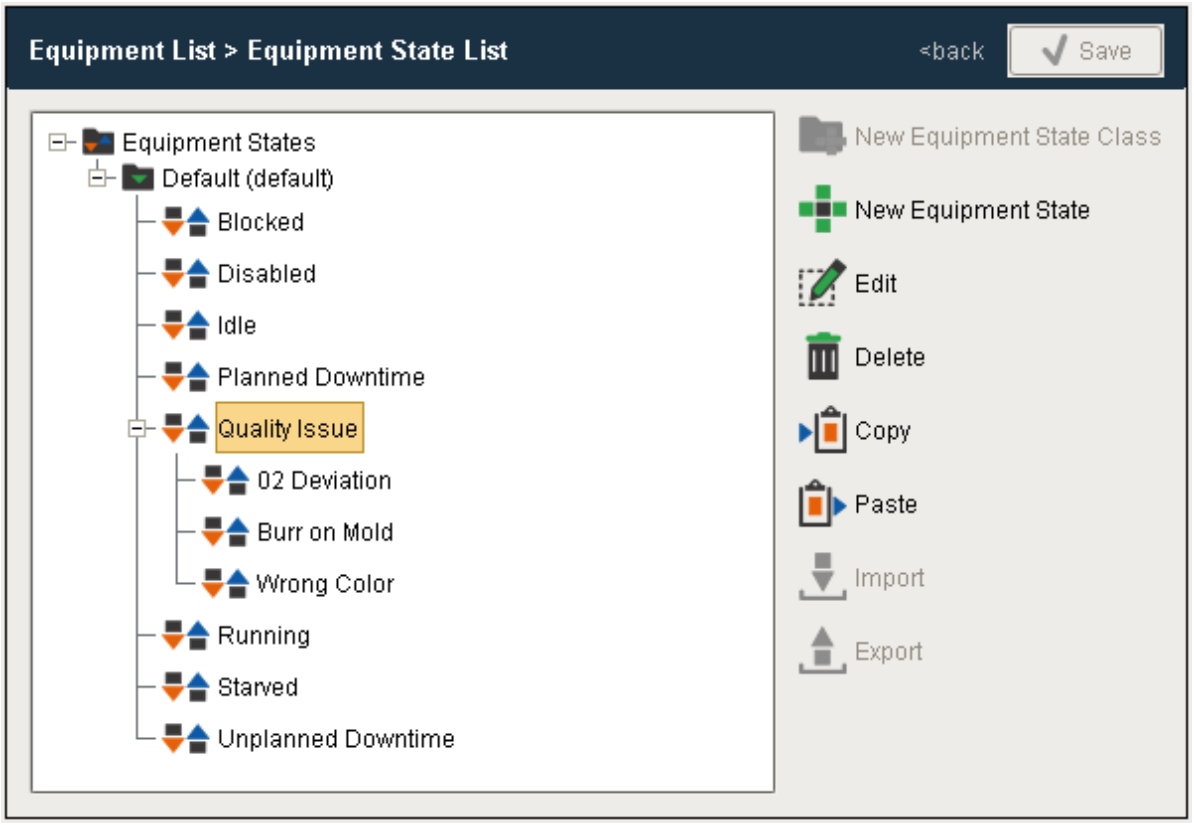
- Detected Equipment State - operator can only select from list of equipment states for the equipment (cell)
- Any Equipment State - operator can select from list of equipment states for any piece of equipment (cell) on the Line
- Sub State - operator can only select from list of sub states created under this Equipment State

Sub States

It is possible to create Equipment States under Equipment States. These are known as Sub States and can be used to group States together that can be used with the Scope Function to allow operators to select specific states (or Downtime reasons) based on the original equipment state.

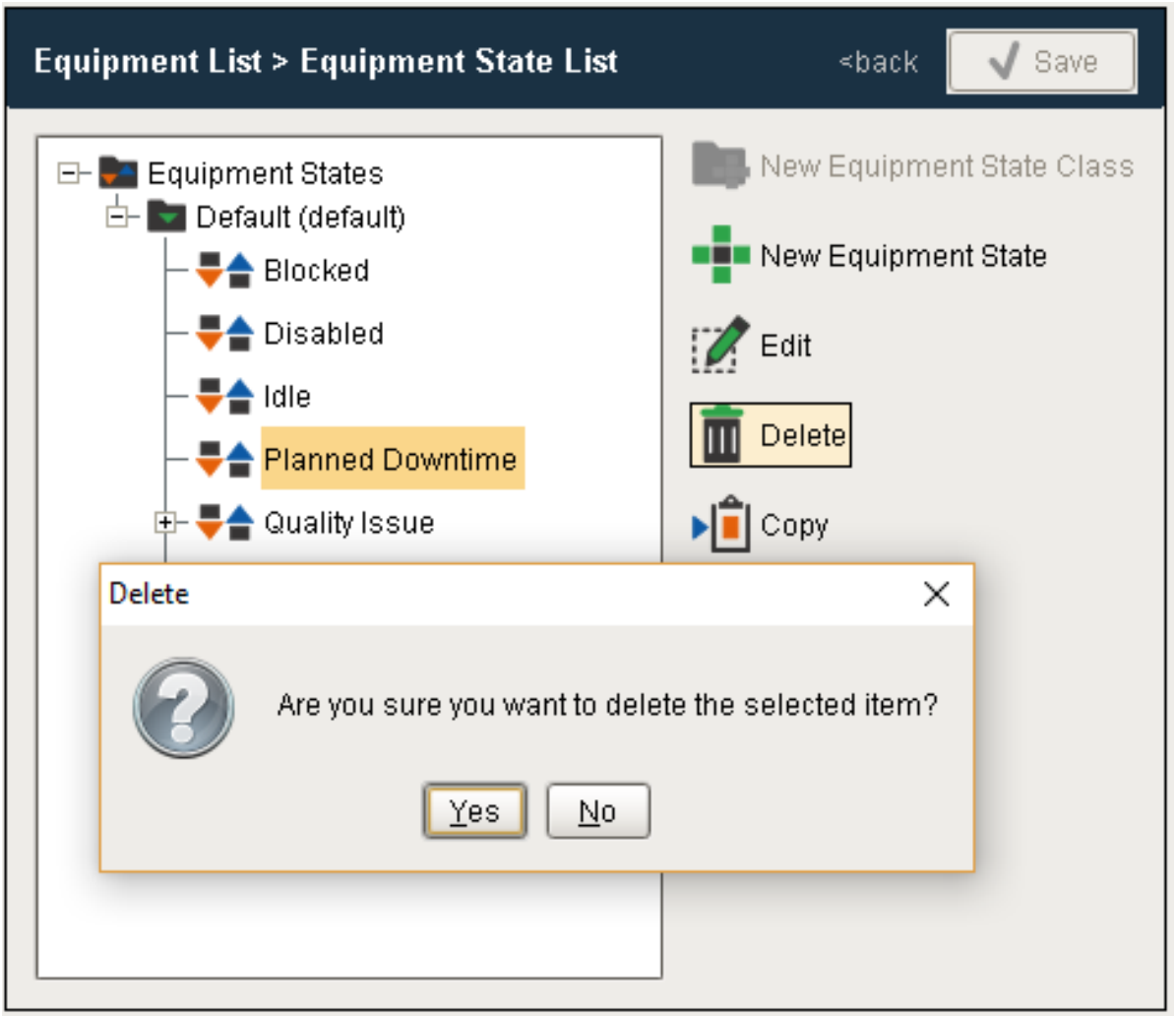
Example

Equipment state for a stamping Press comes through as 'Quality Issue'. **Scope** for 'Quality Issue' is set to 'Sub-State' and **Override** is set to 'Required'. Whenever a 'Quality issue' state occurs on the Stamping Press that causes a Line downtime event, the operator will be prompted and required to select one of the sub-state reasons such as 'O2 Deviation', 'Wrong Color' or 'Burr on Mold'



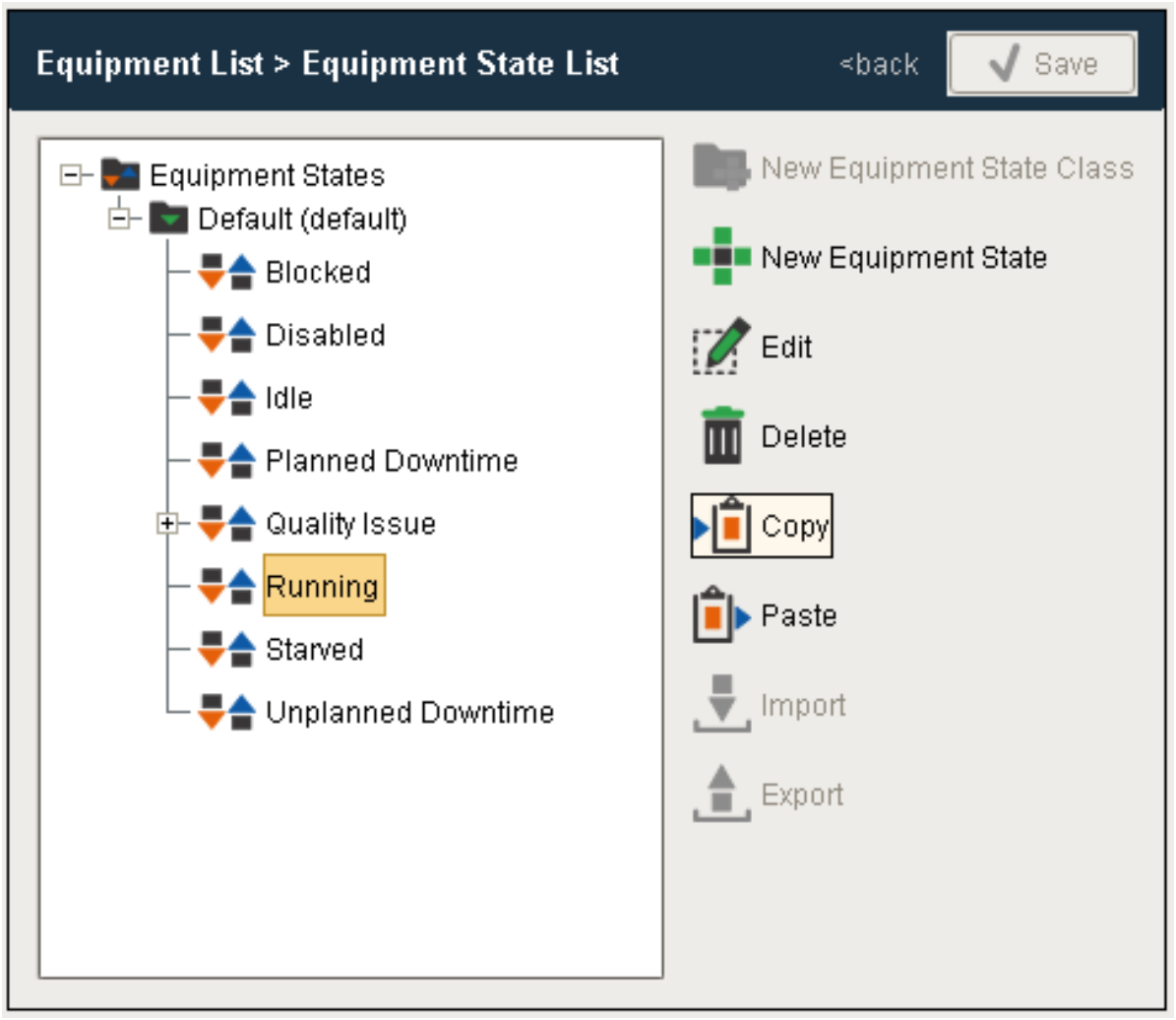
Delete Equipment State

- Click on the Equipment State to be deleted and click **Delete**.



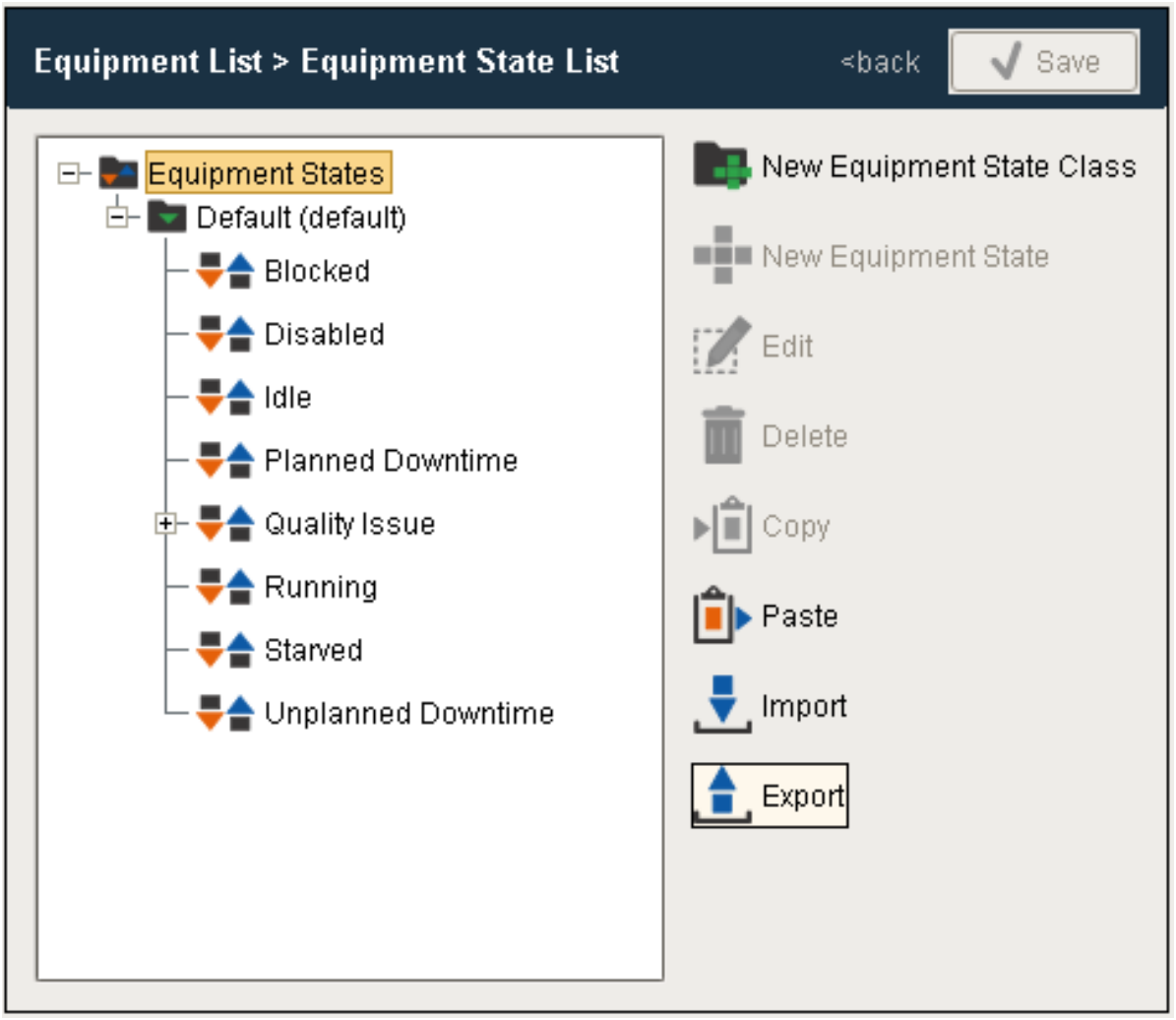
Copy and Paste Equipment State

- Select the **Equipment State** to be copied and click **Copy**.
- Select the **Equipment State Class** to which the state is to be copied to and click **Paste**.
- Expand the destination folder to see the copied equipment state.



Export and Importing Equipment States

- Select the root **Equipment States** to export all Equipment State Classes and States or select a specific Equipment State Class and Click **Export**.
- A file Save dialog box will open allowing you to select where to store the file.
- Select Equipment States or a specific State Class and Click **Import**.
- Select the xml file to be imported and Click **Open**.



Script Functions Associated with Equipment Configuration

This section includes references and examples of using script function to make configuration changes to equipment.

system.mes.getCurrentEquipmentStates

system.mes.getEquipmentModeHistory

system.mes.getEquipmentModeOptions

system.mes.getEquipmentScheduleEntries

system.mes.getEquipmentStateHistory

system.mes.getEquipmentStateOptions

Code Examples

Code Snippet

```
hdr = ['equipPath', 'stateName', 'stateCode', 'stateType']
newData = []


equipPath = '[global]\Nuts Unlimited\Folsom\Receiving\Line 1'

if equipPath != '':
    data = system.mes.getEquipmentStateOptions(equipPath, "", "
")
    for item in data:
        stateName = item.getName()
        if item.getMESObjectType().getName() == 'EquipmentState
Class':
            pass
        else:
            stateCode = item.getStateCode()
            stateType = item.getStateTypeName()
            newData.append([equipPath, stateName, stateCode,
stateType])

eqStates = system.dataset.toDataSet(hdr, newData)
for row in range(eqStates.rowCount):
   for col in range(eqStates.columnCount):
      print eqStates.getValueAt(row, col)
```

Output

```
 [global]\Nuts Unlimited\Folsom\Receiving\Line 1
Unplanned Downtime
3
Unplanned Downtime
[global]\Nuts Unlimited\Folsom\Receiving\Line 1
Planned Downtime
4
Planned Downtime
[global]\Nuts Unlimited\Folsom\Receiving\Line 1
Idle
2
Idle
[global]\Nuts Unlimited\Folsom\Receiving\Line 1
Blocked
5
Blocked
[global]\Nuts Unlimited\Folsom\Receiving\Line 1
Disabled
0
```

```
Disabled
[global]\Nuts Unlimited\Folsom\Receiving\Line 1
Running
1
Running
[global]\Nuts Unlimited\Folsom\Receiving\Line 1
Starved
6
Starved
```

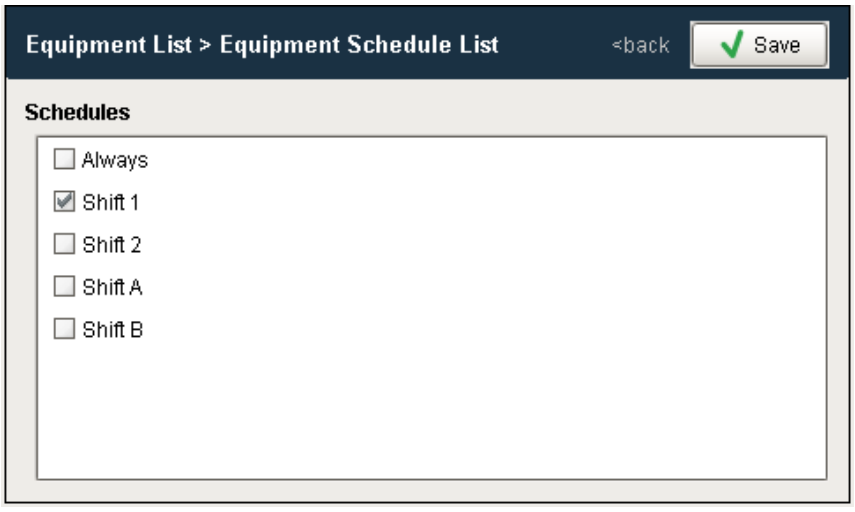
8.1.5 Shift Configuration

For OEE and Production data analysis across shifts, we need to provide a method for defining the current shift at any time during a production run. Shifts are defined at the Line level and enable the Scheduling Engine to accurately estimate how long a work order for a certain quantity of product code should take to complete on a given production line.

Identifying the current Shift can now be tied into the **Ignition Shift Schedule Management System**, where custom shifts can be defined for daily, weekly or rotating schedules. If the Ignition Shift Schedule Management System does not support your needs or is not required, we now provide the hooks so you can tie current production into a custom shift scheduling solution through shift tag collectors in the Production model.

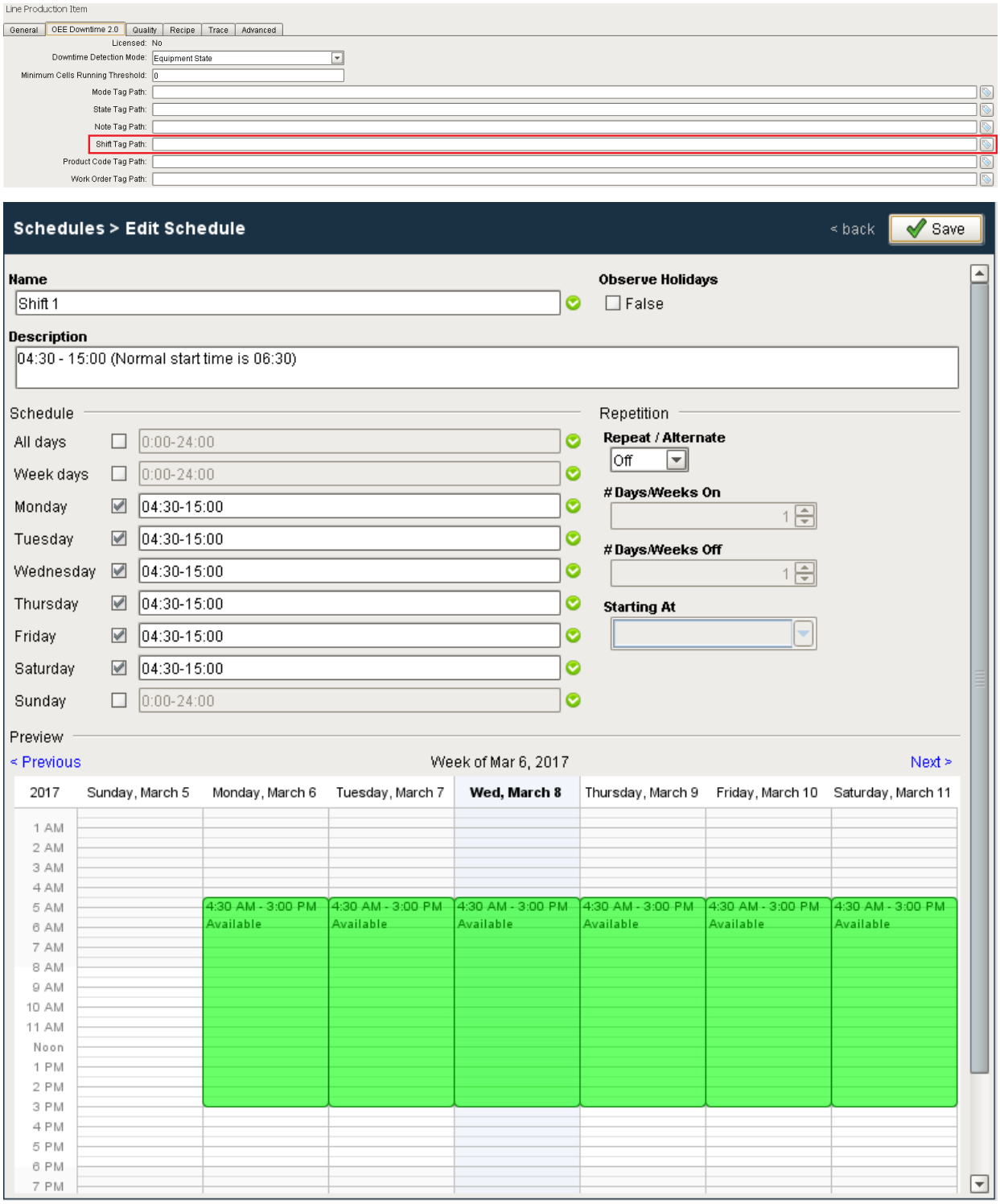
Defining Shifts using the Ignition Schedule Manager

If shifts are defined by using Ignition Schedule Management Component, enabled shifts can be defined for each production line in the **Schedules** menu of the OEE Equipment Manager component as shown below.



OEE Equipment Manager

You can select valid shifts by checking the corresponding box. When implementing shift management this way, leave shift tag collectors at the line and cell level in production model blank. If requiring a custom implementation, use the shift tag collectors in the production model designer to define the current Shift.



Ignition Schedule Manager Component

Defining Shift using the Shift Tag Collector

If you choose not to use the Ignition Schedule Management component, you can simply define the current shift by writing to the Shift Tag Collector provided in the production model for the line whenever the shift changes. Refer to Shift Tag Collector for more details.

8.1.6 Product Definition Configuration

The OEE 2.0 Module provides the Material Manager component to help manage product codes and product code line configuration information. Whatever they are called in your company, whether Product Codes, Pack Codes, SKU's, they all represent the products that are manufactured within your facility. In order to track production counts, calculate OEE Metrics and estimate how long it should take, we need to know what products or materials are being processed and the expected production rate that the line should be capable of.

In OEE 2.0, Product codes are known as Material Definitions and the same framework (object model, scripting functions) is used by both the Track & Trace Module and OEE 2.0 Module. This provides a seamless integration between OEE and Track & Trace when you want to provide lot tracking and genealogy as well as production scheduling and OEE analysis.

In this Page

The OEE Module is a standalone application in that everything needed for product code configuration, scheduling production runs, starting runs and analyzing production data is provided. However, many enterprise implementations take account of the fact that ERP systems, Industrial Engineering or Inventory Management Systems also maintain a list of Product Code information. The MES Product Suite coupled with Ignition, provides the ability to use Web Services or middleware table and script functions to create an interface to obtain product code information from other systems. This ensures that product codes are kept up to date and eliminates duplicate data in multiple systems. If more information is available in other systems regarding product code line configuration, that too can be brought over and scripted to provide the information needed by the OEE and Scheduling Engines.

Configuring Product Codes Manually

The Material Manager component allows you to define which product codes can run on which lines. In addition, the settings for a product code may vary depending on the line it is being produced on and those settings can be defined here.

The following section provide details on using the Material Manager to manage Product Code information.

- Creating Materials
- Material Production Settings
- Configure Routing by Material

Configuring Product Codes from ERP

For companies that have ERP (Enterprise Resource Planning) or other IT System that has product code information, an interface can be created to pull that data and add or update product code configuration settings automatically from it. The actual type of interface created will be dependent upon the supported and preferred method(s) of the ERP / IT administrators. The Sepasoft MES Product Suite provides a Web Services module that can consume SOAP and RESTful API's, but built-on Ignition means that we have many ways that we can interface to access the data. Middleware tables tends to be a common choice providing a neutral method for the dissemination of information between multiple systems without creating any binding tie that is specific between those systems. CSV and flat file can also be used, however if this is the best IT that can be employed to create a robust and reliable interface, then perhaps implementing MES is not for you.

For more information, see our knowledge base article on Creating an Information exchange between MES and ERP.

The system.mes namespace provides a set of script functions that can be used to take information passed from a third party system to create the necessary materials and configuration data.

For more information, refer to our knowledge base article on Creating Material and Supplemental Equipment through Scripting.

Creating Materials

Both OEE 2.0 and Track & Trace use the concept of Material Classes and Material Definitions to manage Product Codes. Material Classes provide a method for grouping similar products together into a category.

Material classes are used extensively in the Track & Trace module to provide production control on which materials can be used or produced by an operation, which lines can process them and where they can be stored. As an example, an operation that packages beer into cans could be constructed to accept any Material that belongs to the Material Class - 'Beer' or it could be limited to a specific Material Definition - 'Firehouse IPA'. In OEE 2.0, Material Classes do not come into play so much, however if your implementation includes Track & Trace or you plan on implementing Track & Trace at a later stage, setting up your Material Classes and Definitions correctly is an important step in the implementation.

The OEE Material Manager component is used to create new material definitions and classes.

In this Page

Material Classes

Adding, Editing and Deleting Material Classes

Material Classes can be created at the **Material Root** level or underneath an existing **Material Class**

- **Add** - Click on Material Root or the Material Class you wish to create a Material Class under and select **New Material Class**
- **Edit -** Click on the Material Class you wish to edit and select **Edit**
- Provide a name for the Material Class and click **Save**.

- **Delete** - Click on the Material Class you wish to delete and select **Delete**

Copying and Pasting a Material Class

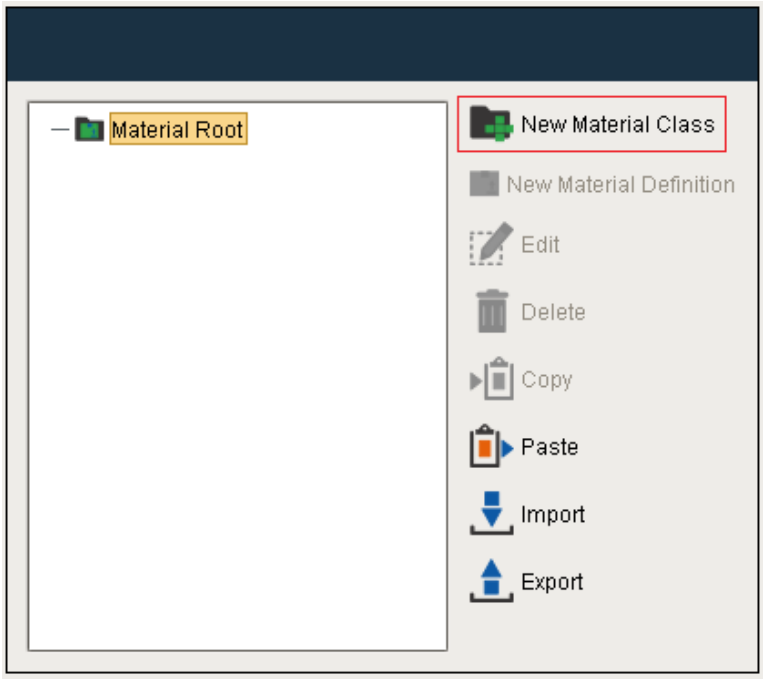
- Select the Material Class to be copied and click **Copy**.
- Select the Material Root or Material Class to copy to and click **Paste**.
- Expand the destination folder to see the pasted Material Class.

Exporting Material Classes

- Select the Material Root or Material Class to be exported and Click **Export**.
- When the save window appears, name the file to be exported and Click **Save**.

Importing Material Classes

- Select the material Root or Material Class to import to and click **Import**.
- Use the File open dialog box to select the xml file to be imported and Click **Open**.

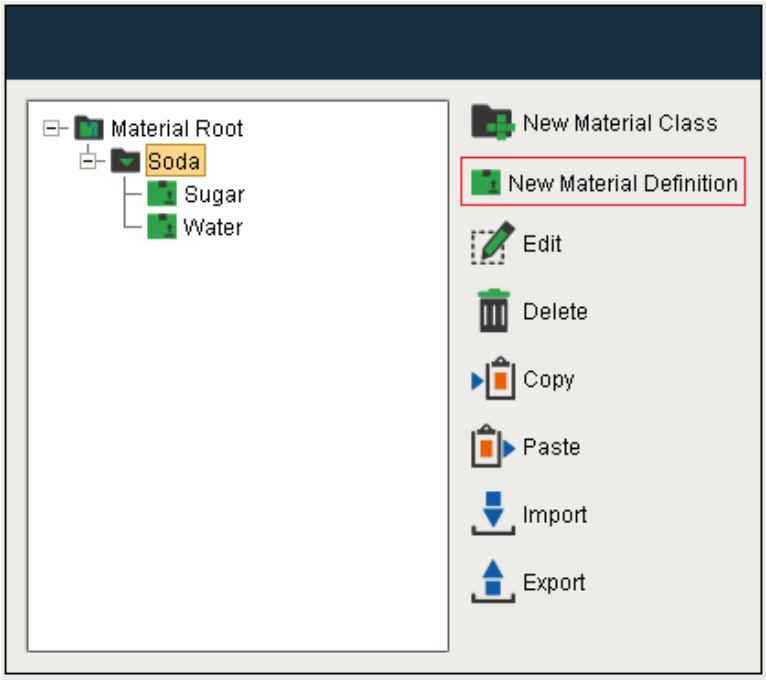


Material Definitions

Material definitions can be added to a Material class but not to the Material Root.

Adding, Editing and Deleting Material Definitions

- **Add -** Click on the Material Class you wish to add a Material Definition to and select **New Material Definition**.
- **Edit** - Click on the Material Definition you wish to edit and select **Edit**.



- Enter a name for the Material Definition.
- Select the Production Lines from the Material Production settings panel that can run this material.
- There are quite a number of options available for Changeover and Production Settings, which we will go into more detail on the Material Production Settings page.
- Click **Save** to add this Material Definition.

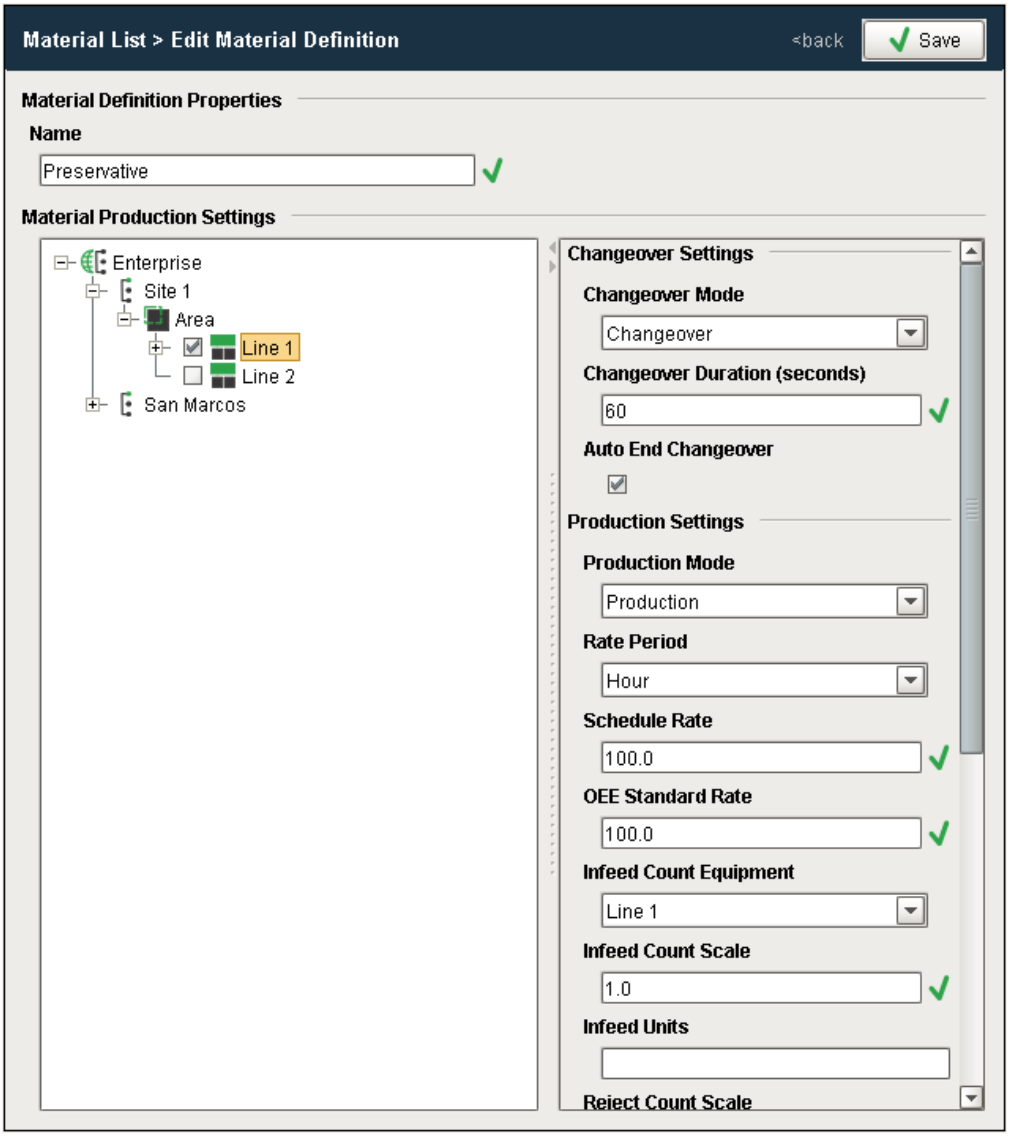
  In the background the Process Segments, Operations Definition and Operation Segments required for this material to run on this line are automatically created for you. In the example, the Operation ***Preservative-Enterprise:Site1:Area:Line 1*** *will be created.*

- ***Delete -*** *Click on the Material Definition to be deleted and select* ***Delete****.*

Copy and Paste Material Definition

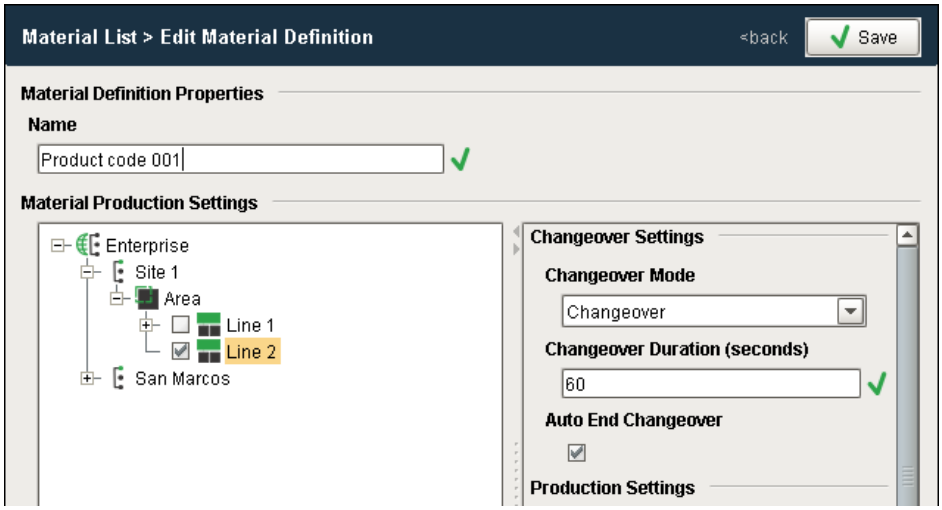
- Select the Material Definition to be copied and click **Copy**.
- Select the Material Class to which the definition is to be copied to and click **Paste**.

- Expand the destination folder to see the pasted Material Definition.



Material Production Settings

When a Material has been configured to run on a production line, there are a number of settings that can be defined that determine how OEE data is captured. These production and changeover settings are set through the OEE Material Manager component.



OEE Material Manager

In this Page

Changeover Settings

The Changeover Settings panel allows you to define what mode the line goes into at the beginning of a production run.

Changeover Mode

Any of the default Equipment Modes (Maintenance, Changeover, Disabled, Production, Other) or custom modes that you have created are valid options. Modes have options for whether production counts are captured or included in OEE. You can use this mode to determine if you want to capture counts, i.e. for setup scrap, but not include it in OEE metrics for this production run.

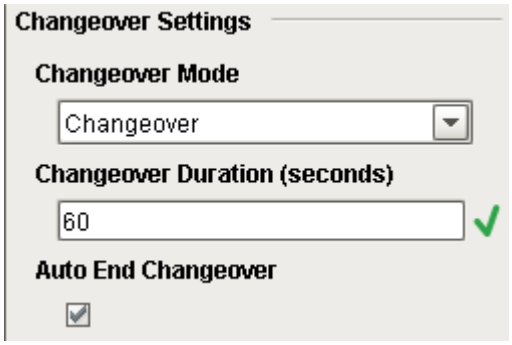
If you have provided a tag for the tag collector path in the OEE Downtime 2.0 Tab, the selected Mode is written to that tag.

Changeover Duration (seconds)

Duration in seconds for the scheduled changeover until this run may start.

Auto End Changeover

Set this to **True** to end the changeover automatically after the scheduled changeover duration. If this is set to false, the selected Changeover Mode will persist until such a time as the tag value provided for the Mode tag collector path in the OEE Downtime 2.0 Tab is written to.



Production Settings

Production Mode

Any of the default Equipment Modes (Maintenance, Changeover, Disabled, Production, Other) or custom modes that you have created are valid options. Modes have options for whether production counts are captured or included in OEE. You can use this mode to determine if you want to capture counts or obtain OEE metrics for this type of run, i.e. **New Product Introduction** or **Testing**.

Changeover Overrun

If you want to automatically capture 'Changeover Overrun' as an OEE metric, this can be achieved in the following way...

1. Create a Mode called **Changeover Overrun**
2. Set **Auto End Changeover** to **True**
3. Set the **Production Mode** to your **Changeover Overrun** mode.
4. When Production actually starts, set the **Mode** Tag provided to the **Mode Tag Collector Path** to the value configured for **Production Mode**.

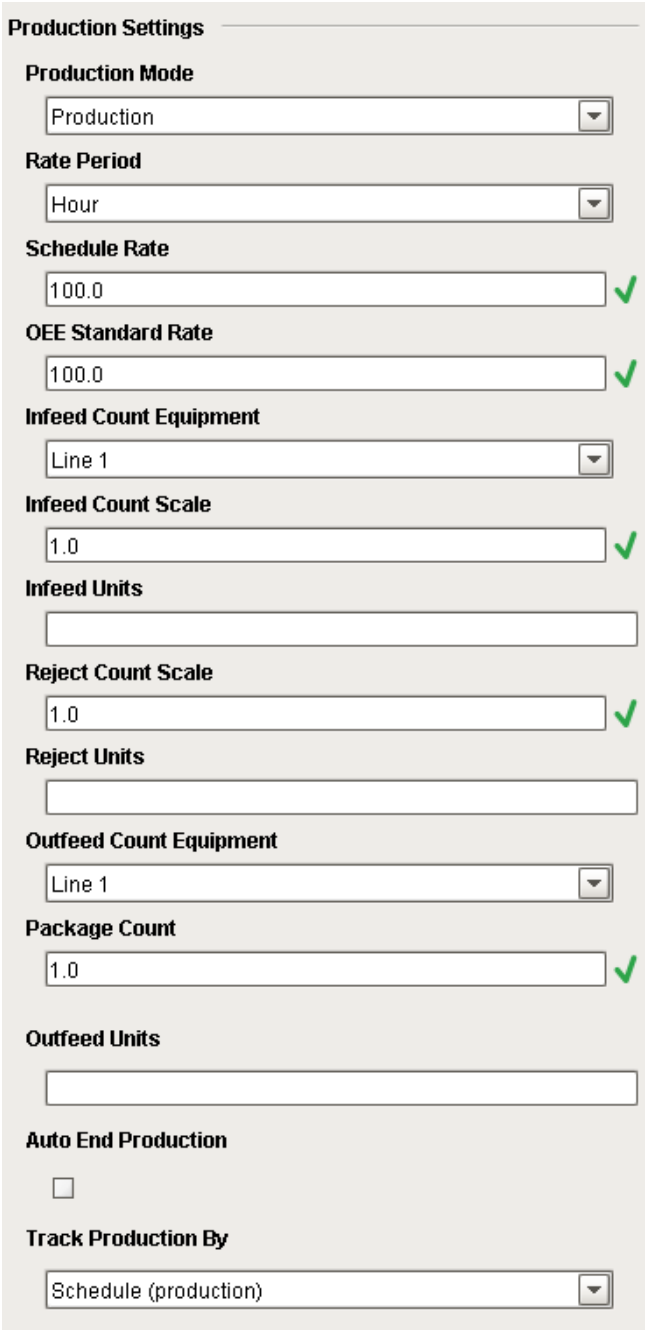
Rate Period

The period of time that applies to all rate values (Standard and Schedule Rate). Valid options are **Hours** or **Minutes**.

Schedule Rate

The realistic production rate that this line can be expected to produce at.

Accepting that there will be some downtime on a line and a certain amount of rework or scrap, the Schedule Rate can be set to be the Standard Rate * Historical OEE of this line to provide a more realistic estimation of how long it will take to complete a production run. The Schedule Rate is used by the Scheduler to estimation completion time based on required quantity.



OEE Standard Rate

The standard rate defines the number of units that theoretically can be processed when the line is running and is used internally to generate a Standard Count value that depicts how many units should have been processed for the amount of time the line has been running. The Standard Count is used to calculate the OEE Performance metric.

When setting up the Standard Rate value, it should be based on the infeed units, whether the infeed count is used or not. Consider the example when the infeed count is in bottles and outfeed count is in cases with a package count of 10 bottles per case. If the OEE Standard Rate is set to 1000 per hour, 1000 bottles must have been counted at the infeed after one hour of runtime for OEE performance to equal 100%. Alternatively, if no infeed count was provided, a combination of 90 cases of good product and 100 rejected bottles must be counted after one hour of runtime, to also equal 100%.

It is possible to configure a production line with a number of combinations of Infeed, Outfeed and waste count tags. The OEE Performance calculation will be based on which tags are provided.

Infeed	Outfeed	Waste	OEE Performance Calculation Based on provided tags	Comment
✗	✗	✗	n/a	
✗	✗	✓	= (Waste Count * Reject Count Scale) / Standard Count	
✗	✓	✗	= (Outfeed Count * Package Count) / Standard Count	
✗	✓	✓	= ((Outfeed Count * Package Count) + (Waste Count * Reject Count Scale)) / Standard Count	
✓	✗	✗	= (Infeed Count * Infeed Count Scale) / Standard Count	
✓	✗	✓	= (Infeed Count * Infeed Count Scale) / Standard Count	
✓	✓	✗		

Infeed	Outfeed	Waste	OEE Performance Calculation Based on provided tags	Comment
			= (Infeed Count * Infeed Count Scale) / Standard Count	
✔	✔	✔	= (Infeed Count * Infeed Count Scale) / Standard Count.	If no infeed count tag is provided or the infeed count value does not change during a production run, the outfeed and waste count values will be used.

Infeed Count Equipment

The equipment that provides the infeed count for the line. Available options are the line itself or any cell on the line. The Infeed Count value comes from the tag associated with the line or cell as defined in the Production Model Designer.

Infeed Count Scale

The Infeed Count Scale is a float value that provides a mechanism for scaling the value of the infeed count tag to the actual number of infeed units.

Imagine a scenario where we are counting strokes on a stamping press for our infeed count, but the number of parts created by the stroke is dependent upon the die set in the press or the product that is being made.

Example

Infeed Count Scale is set to 5. For each infeed count, the number of infeed units recorded by the OEE module will increment by 5

This is similar to the Reject Count Scale for the Reject Count

Infeed Units

The units for the infeed count can be specified here. This could be Cans, Bottles etc.

Reject Count Scale

The Reject Count Scale is a float value that provides a mechanism for scaling the value of the reject count tag to the actual number of reject units created.

Imagine a scenario where the reject count is for a pallet or case of product, but the number of parts on the pallet or case is dependent upon the product that is being made.

Example

Reject Count Scale is set to 24. For each reject count, the number of rejected units recorded by the OEE module will increment by 24

This is similar to the Infeed Count Scale for the Infeed Count

Reject Units

The units for the reject count can be specified here. This could be Cans, Bottles, Cases, Pallets etc.

Care needs to be taken on how waste is counted. By default waste is considered to be in the same units as the infeed count, so in OEE calculations, it is divided by the Package Count. If the waste value provided is in fact in the same units as the outfeed count units (cases, for example) then the waste count must be multiplied by the same value as the package count. The reject count scale setting can be used to handle this.

Outfeed Count Equipment

The equipment that provides the outfeed count for the line. Available options are the line itself or any cell on the line. The outfeed count value comes from the tag associated with the line or cell as defined in the Production Model Designer.

Package Count

The **Package Count** is a float value that provides a mechanism for associating the value of the outfeed count to the value of the infeed count. It is only used internally to calculate OEE values and does not modify the infeed, waste or production counts. Those are recorded as set up.

Examples

- Infeed count is in bottles and outfeed count is in cases. For a particular product, there are 24 bottles per case. For this example, we would set **Package Count** to 24.
- Infeed count is in lbs and outfeed count is in cans. For a particular product, there are 12oz in each can. For this example, we would set **Package Count** to 0.75.
- Infeed count is in lbs and outfeed count is in cans. For a particular product, there are 22oz in each can. For this example, we would set **Package Count** to 1.375.

Care needs to be taken on how waste is counted. By default waste is considered to be in the same units as the infeed count, so in OEE calculations, it is divided by the Package Count. If the waste value provided is in fact in the same units as the outfeed count units (cases, for example) then the waste count must be multiplied by the same value as the package count. The reject count scale setting can be used to handle this.

Outfeed Units

The units for the outfeed count can be specified here. This could be Cans, Bottles, Cases, Pallets etc.

Auto End Production

When set to **True**, if **Track Production By** is set to **Schedule (production),** the production run will automatically be ended once the desired number of units have been produced. If **Track Production By** is set to **Schedule (time),** the production run will automatically be ended once the elapsed time is equal to the scheduled time.

When set to **False**, the run must be ended using the Run Director or Schedule Selector components, or through scripting functions.

Track Production By

Setting this to **Schedule (production)** will track the production by the scheduled production rate. Setting the property to **Schedule (time)** will track the production by scheduled time.

The MES Schedule View Component has a progress bar that will update according to the **Track Production Setting**. If set to **Schedule (production)**, the progress bar will display as a percentage of current outfeed / Scheduled Qty. The Schedule estimated completion time will update every minute to extend the duration of the scheduled run if production is falling behind schedule. If set to **Schedule (time)**, the progress bar will display as a percentage of Elapsed Time / Scheduled Time.

Valid options are:

- Schedule (production)
- Schedule (time)

If you want the scheduler to update the estimated completion time at an interval greater than a minute, this can be achieved using the MES Object Editor to change the Update Interval for the operation.

Programmatically Change Production Settings

Settings may be changed through the Material Manager component in the designer. However, often changes should be driven programmatically, for instance for ERP integration. Here is an example of setting the OEE Rate (a.k.a. Standard Rate) through scripting.

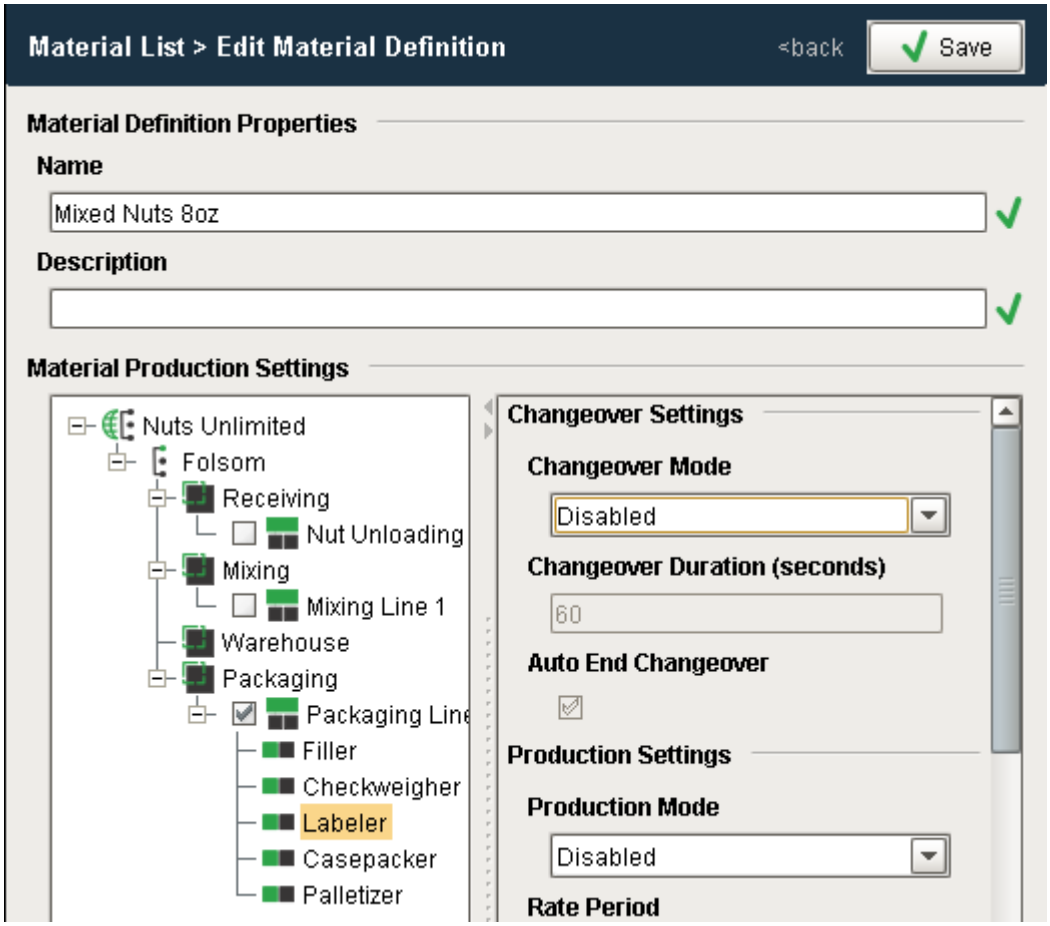
Changing Production Settings Programatically

```
##Load the Operation Definition based on the Operation Definition
Name
opDefName = 'PC01-Enterprise:Site:Area:Line 1'
obj = system.mes.loadMESObject(opDefName, 'OperationsDefinition')


##Load the Operations Segment for Production
opSegUUID = obj.getComplexProperty('SegmentDependency', 'Productio
n Dependency').getSegmentRefUUID()
opSeg = system.mes.loadMESObject(opSegUUID)


##Iterate through the Production Settings for the Operation
Segment.
##For each production setting (i.e. for each line, cell, cell
group, etc.) set the OEE Rate
count = opSeg.getComplexPropertyCount('ProductionSettings')
for i in range(count):
    prodSet = opSeg.getComplexProperty('ProductionSettings', i)
    prodSet.setOEERate(78.6)
    opSeg.setPropertyValue('ProductionSettings', prodSet)


##Save the Operation Segment to make the changes manifest
system.mes.saveMESObject(opSeg)
```

Configure Routing by Material

When configuring material to run on a production line, it may be that not all process cells in a line will be used for certain products. An example could be a packaging line where a certain product does not requires labels to be attached from the labeler. In this case we can configure the labeler cell to be disabled whenever this product code runs on the packaging line.

The OEE Material Manager allows us to do this by setting the Production Mode to DISABLED for the labeler cell in the Material Production Settings Pane.



If we want to display to an operator the routing of which cells need to operational during a production run, we can use scripting to access the production settings for this operations segment.

Example 1

```
segName = 'Mixed Nuts 8oz-Nuts Unlimited:Folsom:Packaging:
Packaging Line 1'
mesObject = system.mes.loadMESObject(segName, "OperationsSegmen
t")
prodList = mesObject.getComplexPropertyItemNames('ProductionSet
tings')
for item in prodList:
    print item, " - ", mesObject.getComplexProperty('Production
Settings',item).getModeRef().getMESObject().getModeType().
getName()
```

Output

```
Packaging Line 1 - Production
Packaging Line 1:Casepacker - Production
Packaging Line 1:Checkweigher - Production
Packaging Line 1:Filler - Production
Packaging Line 1:Labeler - Disabled
Packaging Line 1:Palletizer - Production
>>>
```

Example 2

```
matName = "Mixed Nuts 8oz"
lineName = "Packaging Line 1"
cellName = "Casepacker"

for index in range(count):
    prop = seg.getComplexProperty("ProductionSettings", index)
    if prop.getName().endswith(cellName):
        modeRef = prop.getModeRef()
        mode = modeRef.getMESObject()
        modeType = mode.getModeType()
        print modeType.getName()
        break;
```

Output

```
Production
>>>
```

8.1.7 Production Scheduling & Dispatch

ISA-95 defines Detailed Production Scheduling as

'...*the collection of activities that take the production schedule and determine the optimal use of local resources to meet the production schedule requirements*'.

The Sepasoft MES solution provides both the OEE 2.0 and Track & Trace modules that can be used to build your Detailed Production Scheduling system.

Track & Trace Module

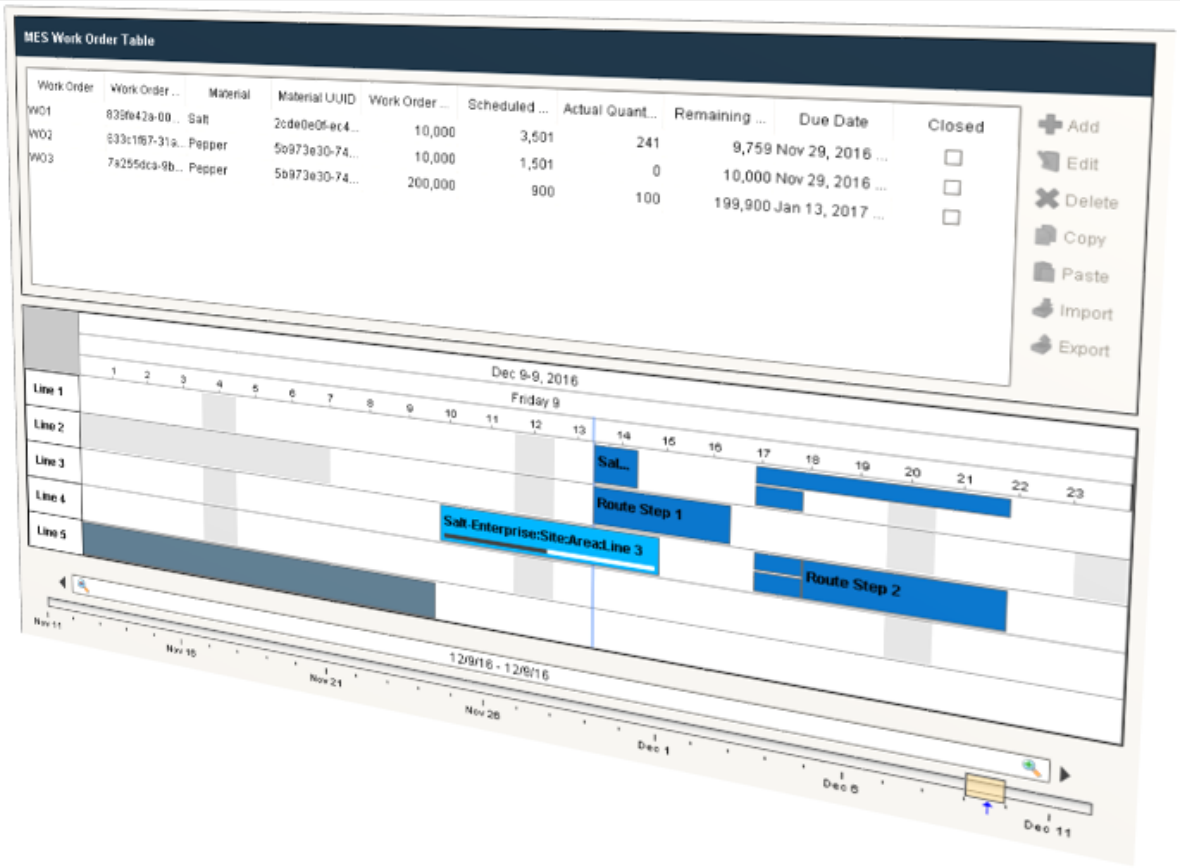
The Track & Trace module can be used for detailed production scheduling on any type of manufacturing operation. Refer to Operations Scheduling for more details.

OEE 2.0 Module

The OEE 2.0 module can also be used for detailed production scheduling of OEE production runs, maintenance and cleaning operations. Refer to Operations Scheduling for more details.

The MES Scheduler is common between both Track & Trace and OEE 2.0.

MES Scheduler



The MES Scheduler provides finite scheduling functionality that seamlessly integrates with the OEE 2.0 and Track & Trace modules. When combined with Ignition, these scheduling features allow operations to easily adapt to last minute or frequent changes that commonly occur in production environments. This is accomplished by monitoring production in real-time, handling delays, production routes, scheduling changes, notification of production priority changes and more.

Most Manufacturers rely on ERP and Inventory Management Systems to handle the complex process of inventory planning, high level Customer Order scheduling, routing, transportation logistics and accounting. However when it comes to the detailed planning and scheduling of shift personnel, production lines and maintenance activities, this tends to occur at the MES level. If scheduling information and production performance data is stored in separate systems, whether in spreadsheets or stand-alone scheduling software, that cannot be accessed and combined, we're missing the opportunity to provide powerful insight into operational activities and production line utilization.

The Scheduler provides finite scheduling functionality at the MES layer that allows operations, maintenance and planners to create a detailed web based schedule that can pull work orders directly from ERP, allowing them to be modified at the MES level to

account for last minute changes, and shared with everyone within the organization. With our direct connection to actual production line status and counts, the schedule can provide real-time status monitoring of actual vs scheduled production, automatic handling of delays and updates of inventory consumption and order fulfillment data back to ERP, as well as providing schedule adherence analytics to help drive continuous improvement initiatives.

With flexibility in mind, work orders and production schedule entries can be created in the following ways:

- From Customer Orders or Schedule Entries in ERP or other scheduling software
- Manually created using the Work Order Table and Line Schedule View components
- Dynamically generated using scripting
- Imported from other sources

Features

- **Stand-Alone Scheduling or ERP Work Order Integration**
- **Simple 'Drag & Drop' Work Order Scheduling**
- **Production Routing**
- **Shift Scheduling**
- **Automatic Adjustment for Production Delays**
- **Finite Scheduling estimates completion date based on constraints**
- **Real-time production progress**
- **Seamless integration with all other Sepasoft MES modules**
- **Visual Scheduler for easy communication of schedule between departments**
- **Overlapping schedule entries**
- **Auto Extend Scheduled Run (when production is behind schedule)**

In This Section

8.1.8 Production Order Execution

Sepasoft's OEE 2.0 module provides a number of components and scripting functions that allow you to create an operations management framework that can handle how your enterprise functions. When coupled with the Track & Trace module, the number of components and scripting functions available are extended to allow for the management and execution of OEE Production runs as well as non production operations.

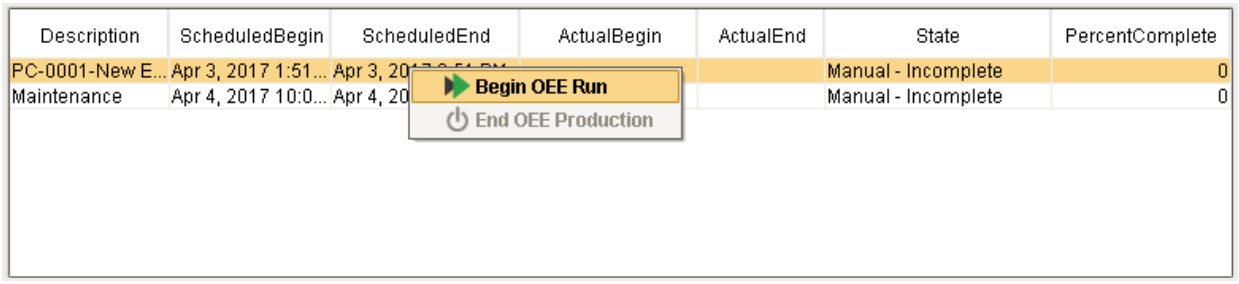
In this section, we will explore how to execute and control production runs and handle running changeovers with multiple products on a line.

In this section

Operations Control

Components

If an OEE run or operation has been scheduled, it can be started and ended using the MES Schedule Selector component. Right click on the desired operation and select **Begin OEE Run** for an OEE Run or **Begin Operation** for a non OEE Run operation.



MES Schedule Selector

The OEE Run Director component can also be used to start and end OEE runs, whether they are scheduled or un-scheduled. It cannot however be used to control un-scheduled non OEE operations. That functionality comes as part of the Track & Trace module and can be implemented using the MES Operation Selector and MES Segment Selector components.

Run Director Component

Automated Schedule Control

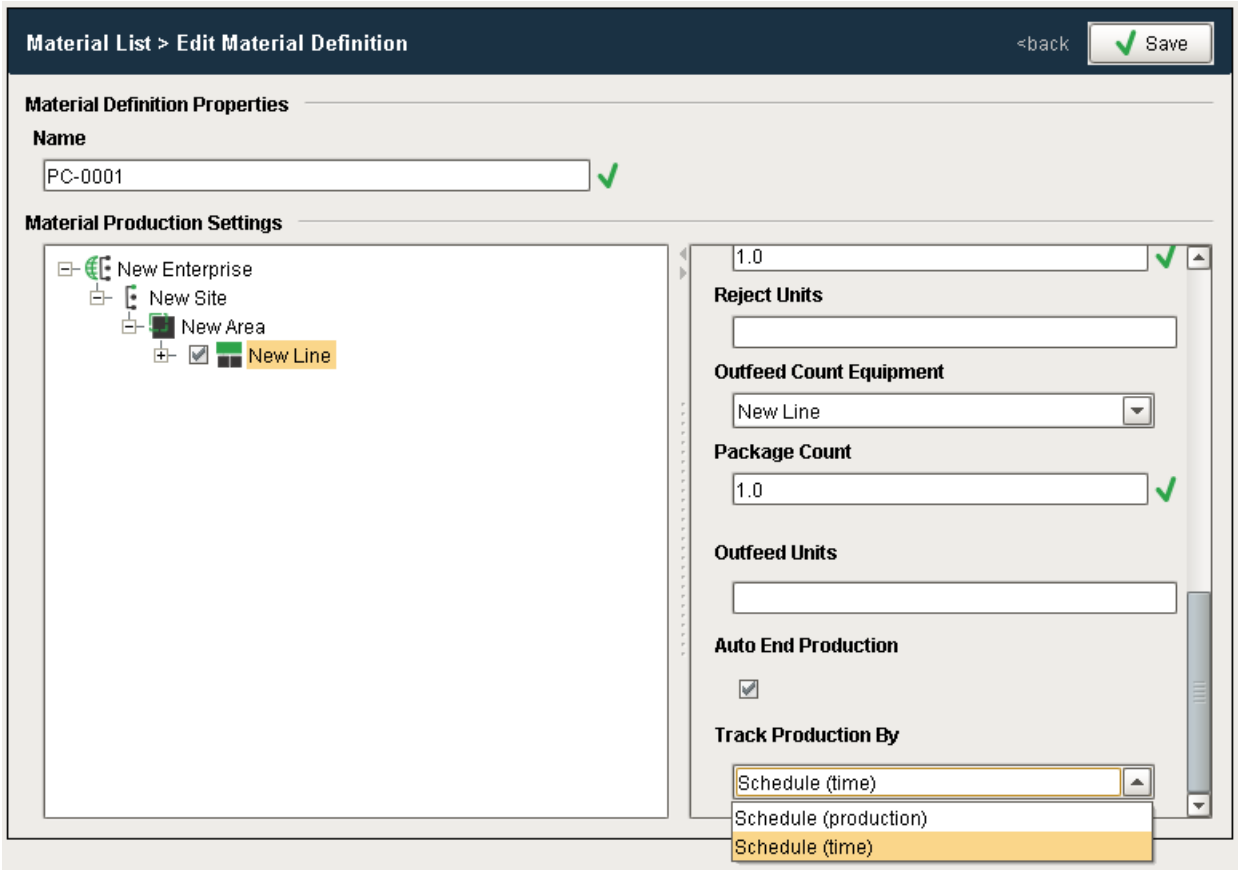
OEE runs can be configured to automatically start and end based on the schedule time and production count.

To setup a run to automatically start, the Operations Definition that was created for you for the product code / line combination can be modified in the MES Object Editor. Under **Trigger Operation Begin,** change the **Mode** to **Schedule (time)** and check Auto.

This same functionality can also be achieved by adding a script to the **BeginSchedule** event for the OperationsRequest in the MES Object Events section of the Production Model Designer at the Enterprise Production Item level.

Trigger Operation Begin Section of the Operations Definition

To automatically end an OEE run, set **Auto End Production** to True and set **Track Production By** to **Schedule (time)** to stop the OEE Run based on the schedule, or to **Schedule (Production)** to automatically end the run after the required quantity has been produced.



Setting Auto End Production in the Material Manager

Scripting Functions

The following scripting functions are provide to control OEE Runs...

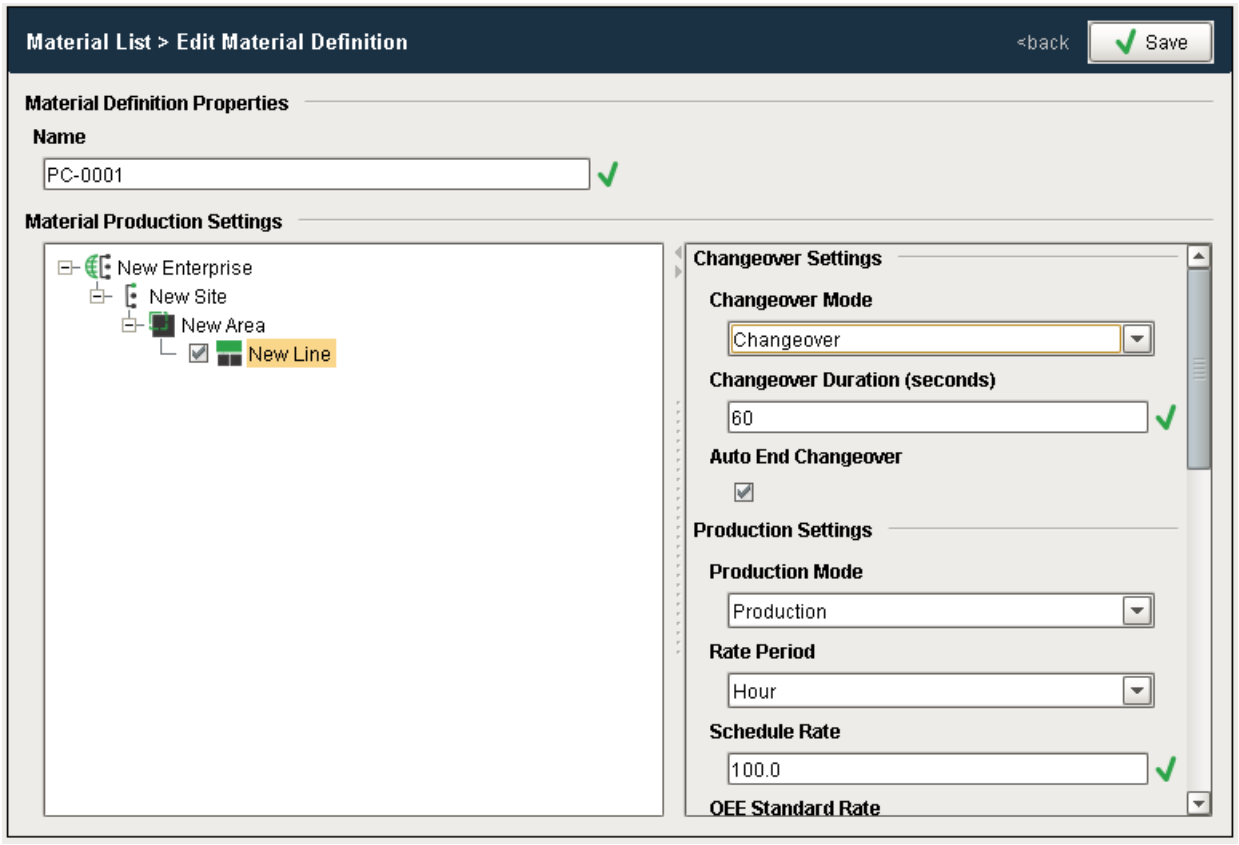
- system.mes.oee.abortRun
- system.mes.oee.beginOEERun
- system.mes.oee.endCellChangeover
- system.mes.oee.endOEEChangeover
- system.mes.oee.endOEEProduction

- system.mes.oee.getOEEActiveSegment
- system.mes.oee.getOEEAllActiveSegments
- system.mes.oee.indexCellProduct
- system.mes.oee.removeMaterialOperationSegments
- system.mes.oee.updateMaterialOperationSegments

Changeovers

When an OEE Run is started, the Line Mode is automatically set to the mode as defined in the OEE Material Manager for the selected product and line combination. In the example screen, the Line mode will be set to **Changeover** for the first 60 seconds of the production run, at which point the mode will then be set to **Production**.

For more information on the Changeover and Production settings including how to set up the system to automatically create a Changeover Overrun condition, please refer to the Material Production Settings help.



MES Material Manager

Running Changeovers

OEE 2.0 provides support for running changeovers, where multiple products can be being processed on the same line at the same time.

When a production run is started, the product code specified for the run will be indexed to all cells on the line. If a new production run that specifies a different product code is then started on the same line, the new product code will be automatically indexed to the first cell on the line, or if the first cell defined in the line is a cell group, all cells (and cell groups) within that cell group will be indexed with that product code. As the new product code makes it way down the production line, the product code can be indexed to each cell either by setting the value of the tag bound to the cell Product Code Tag Path to the product code, or by using the function system.mes.oee.indexCellProduct. This function can be called sequentially for each cell on the line. When this function is called, all upstream cells will have their product code indexed too if they have not already been indexed.

8.1.9 OEE Production Data Collection

The type of production analysis that can be performed will be based on the production data that is collected. In OEE 2.0, production data is captured 24/7 regardless of whether a run is scheduled or started in the MES application. The source of the data collected is dependent upon how it has been configured.

Shift Information

Shift information is automatically derived based on the shift configuration for the production line. Refer to Shift Configuration for more details.

Production Counts

Production counts are captured through the use of MES counters and can be collected for the line and cells within the line. Refer to MES Counters for more details.

Equipment Mode and Status

The status of a production line and cells within that line are captured through a tag collector path configured in the production model. Refer to Setting Up Equipment Modes and Setting Up Equipment Modes for more details.

Schedule Information

Information about scheduled runs is automatically stored whenever the OEE Run Director or MES Schedule View component is used to control a scheduled run.

Downtime Information and User Notes

The OEE Downtime Table provides a method for a user to modify the cause of line and cell downtime events on a line, and allows for operator notes to be entered regarding downtime events. Downtime notes can also be entered by providing a tag to the downtime note tag collector in the production model. Refer to Downtime Note for more details.

Additional Factors

Additional factors allow you to capture any other type of data that you wish to analyse along with the production data. Refer to Additional Factors for more details.

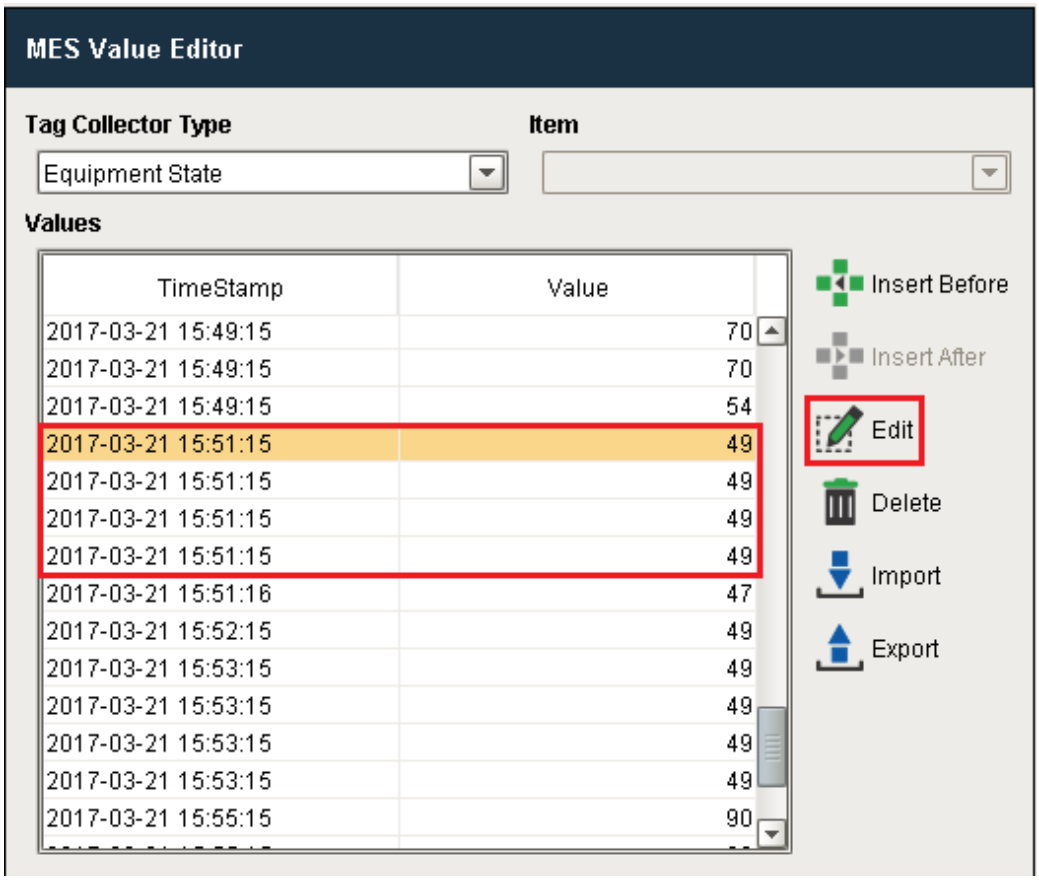
Configuration Information

Data regarding the equipment or material configuration, such as downtime detection method, key cell, standard rate etc., is automatically stored from the configuration information setup for the equipment and material. refer to Equipment Configuration for more details on how to configure equipment. Refer to Product Definition Configuration for more details on how to setup materials.

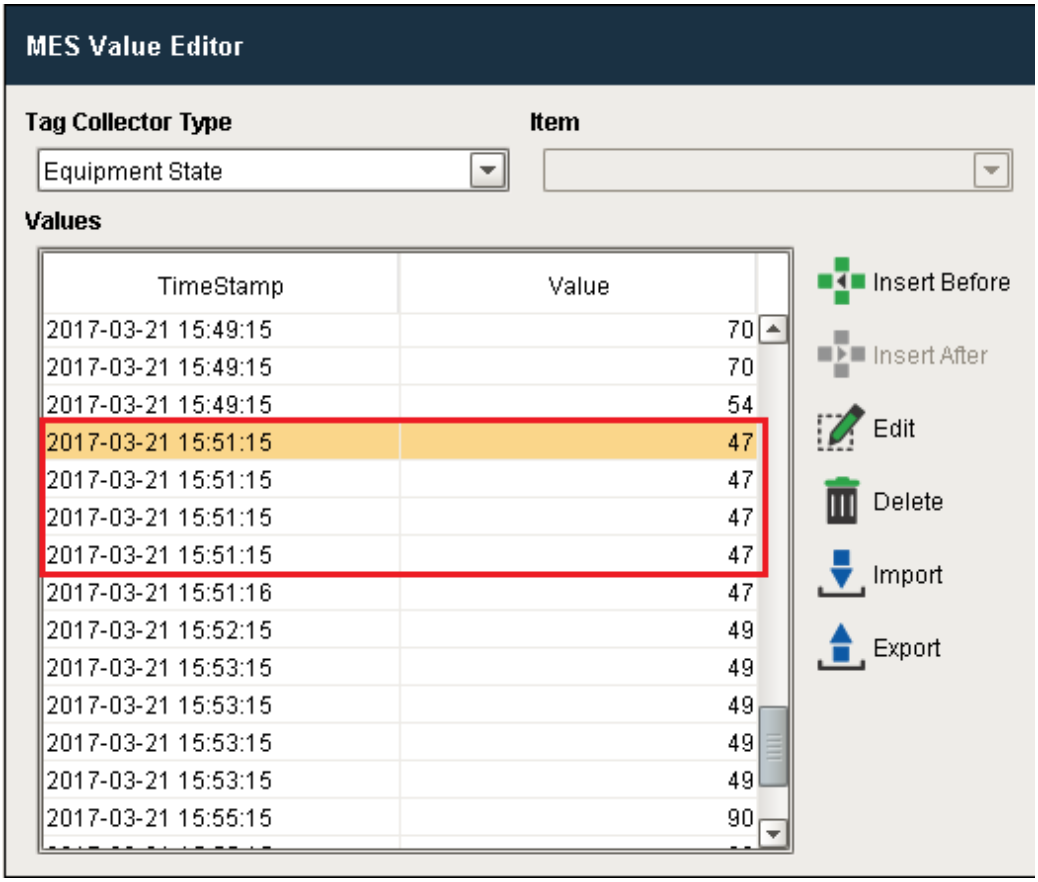
8.1.10 Adjusting Production Run Data

Production values are recorded automatically through values passed via tag collectors, such as counts and status, and those values set by the OEE engine, such as Mode, Standard Rate, shift etc. The MES Value Editor component allows for these collected production values to be edited or deleted and for new values to be inserted. This provides an easy method for correcting production values where the line was set to the wrong mode, line was recorded as running when it was in fact down, production counts are off, or the wrong product code or work order was selected.

In this Section


MES Value Editor
Tag Collector Type
Item
Equipment State
Values
TimeStamp
Value
2017-03-21 15:49:15 70
2017-03-21 15:49:15 70
2017-03-21 15:49:15 54
2017-03-21 15:51:15 49
2017-03-21 15:51:15 49
2017-03-21 15:51:15 49
2017-03-21 15:51:15 49
2017-03-21 15:51:16 47
2017-03-21 15:52:15 49
2017-03-21 15:53:15 49
2017-03-21 15:53:15 49
2017-03-21 15:53:15 49
2017-03-21 15:53:15 49
2017-03-21 15:55:15 90
Insert Before
Insert After
Edit
Delete
Import
Export


MES Value Editor > Edit Value
<back
Save
Value
47



Tag Collector Types

Recall that production values such as equipment modes, states and counts are recorded 24 /7. If the recorded values need to be modified as production counts were off or the line mode was captured as Maintenance when it should have been Production, these scripting functions or the MES Value Editor component, can be used to correct the values.

The Tag Collector Types are used by the MES Value Editor component and the script functions listed below to read and modify production values recorded via tag collector paths and by the OEE engine.

Each tag collector type may have a different datatype and some tag collector types have a **key** (where there is more than one stored value for the tag collector type). Examples of these would be MES Counters, where the Tag Collector Type **Equipment Count** would have the default **Material Out** and any other user added mes counter names. Additional Factors would also use the key to distinguish between the user defined additional factors.

The Equipment State tag collector has an additonal parameter called the **Auxiliary Value**. The getTagCollectorValue() and updateTagCollectorValue() have an overloaded function to handle the auxiliary value name for this tag collector type.

Scripting Functions for Tag Collectors

- system.mes.addTagCollectorValue
- system.mes.addTagCollectorValues
- system.mes.getTagCollectorDeltaValue
- system.mes.getTagCollectorLastTimeStamp
- system.mes.getTagCollectorLastValue
- system.mes.getTagCollectorPreviousTimeStamp
- system.mes.getTagCollectorPreviousValue
- system.mes.getTagCollectorValue
- system.mes.getTagCollectorValues
- system.mes.removeTagCollectorValue
- system.mes.removeTagCollectorValues
- system.mes.updateTagCollectorLastValue
- system.mes.updateTagCollectorValue
- system.mes.updateTagCollectorValues

Tag Collector Type	Data Type	Key	Auxiliary Value	Description
Equipment Additional Factor	String	Name of user defined additional Factors	N/A	See Additional Factors for more details
Equipment Count	Long	Name of user defined MES counters and the default **Material Out**	N/A	Value of the MES Counter as defined in the key. See MES Counters for more details
Equipment Cycle Count	Long	N/A	N/A	See Analysis Datapoints and Settings - Cycle Count for more details

Tag Collector Type	Data Type	Key	Auxiliary Value	Description
Equipment Downtime Note	String	N/A	N/A	Any downtime notes added through the tag collector or entered through the OEE Downtime Table component
Equipment Infeed Count Scale	Float8	N/A	N/A	See Infeed Count Scale for more details
Equipment Infeed Units	String	N/A	N/A	See Infeed Units for more details
Equipment Mode	Int4	N/A	N/A	See Setting Up Equipment Modes for more details
Equipment Operation UUID	String	N/A	N/A	Unique Identifier for currently running operation
Equipment Outfeed Units	String	N/A	N/A	See Outfeed Units for more details
Equipment Package Count	Float8	N/A	N/A	See Package Count for more details
Equipment Product Code	String	N/A	N/A	Product code currently being processed on this equipment
	String	N/A	N/A	

Tag Collector Type	Data Type	Key	Auxiliary Value	Description
Equipment Rate Period				See Rate Period for more details
Equipment Reject Count Scale	Float8	N/A	N/A	See Reject Count Scale for more details
Equipment Reject Units	Float8	N/A	N/A	See Reject Units for more details
Equipment Schedule Count	String	N/A	N/A	Amount of product that should have been produced (Target) based on the schedule rate
Equipment Schedule Duration	Float8	N/A	N/A	Duration of scheduled run
Equipment Schedule Rate	Float8	N/A	N/A	See Schedule Rate for more details
Equipment Shift	String	N/A	N/A	The shift as defined in the Ignition Schedule Management component or passed to the Shift Tag Collector
Equipment Standard Rate	Float8	N/A	N/A	See standard rate for more details

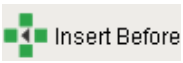
Tag Collector Type	Data Type	Key	Auxiliary Value	Description
Equipment State	String	N/A	EquipmentUUID	The unique identifier for the equipment
Equipment State	Int4	N/A	State	Equipment state value
Equipment State	Int4	N/A	OriginalState	The original equipment state before it was updated
Equipment State	String	N/A	DifferedToUUID	If the original EquipmentUUID is changed using the Downtime Table then the new uuid is DifferedToUUID
Equipment State	String	N/A	DifferedState	If the original state is changed using the Downtime Table then the new state is DifferedState
Equipment Target Changeover Time	Float8	N/A	N/A	Amount of time in minutes set for Target Changeover. See Changeover Duration for more details
Equipment Work Order	String	N/A	N/A	Work order processed on this equipment
Line Infeed Count Equipment UUID	String	N/A	N/A	Unique identifier for the equipment where the line infeed count came from

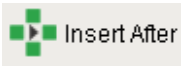
Tag Collector Type	Data Type	Key	Auxiliary Value	Description
Line Outfeed Count Equipment UUID	String	N/A	N/A	Unique identifier for the equipment where the line outfeed count came from

Item

When there are multiple values within a tag collector, this specifies which one to show. As an example, there can be multiple MES counters for the **Equipment Count** tag collector type. If not needed, pass an empty string. In other words it is the name of the MES counter or the name of the Additional Factor.

Insert Value Before or After

- Select the value to be changed.
- Click on the 

button or 

button.
- Set the new time stamp and change the value. Click Save.

Editing and Deleting Value

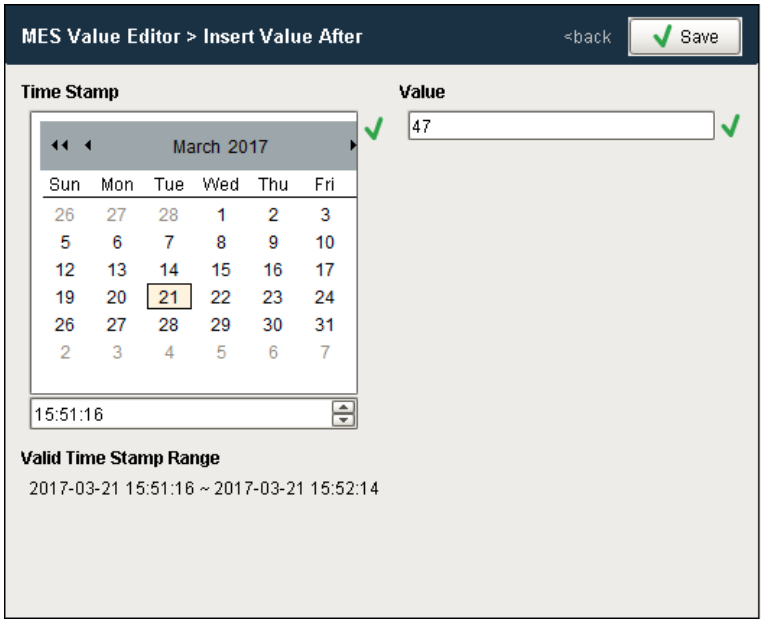
- **Edit -** Click on the value you wish to edit and select **Edit**
- Provide the new value and click **Save**.
- **Delete** - Click on the value you wish to delete and select **Delete**

Exporting Values

- Select the Tag Collector Type to be exported and Click **Export**.
- When the save window appears, name the file to be exported and Click **Save**.

Importing Values

- Select the Tag Collector Type to import to and click **Import**.
- Use the File open dialog box to select the xml file to be imported and Click **Open**.



8.1.11 Production Performance Analysis and Reporting

Access to production information can be obtained in several ways in OEE 2.0. Live Analysis provides a method for accessing current run information via ignition tags. The MES Analysis Selector and MES Analysis Controller components provide a method for obtaining real-time as well as historical run information and this information can also be accessed via the MES Analysis Scripting functions. The OEE Downtime Table provides a view of causes of line downtime as well providing a method for user to modify causes and enter downtime notes, and the OEE Time Chart provides an overall view of the schedule, mode and state changes occurring on a production line and associated cells.

In this section, we will show how to setup Live Analysis, use analysis components and scripting functions, and create reports using Stored Analysis.

Live Analysis

In the OEE 1.0 module, OPC tags were provided by the Production OPC server to provide real-time OEE Run status monitoring. In OEE 2.0 the Production OPC tags have been replaced with Live Analysis.

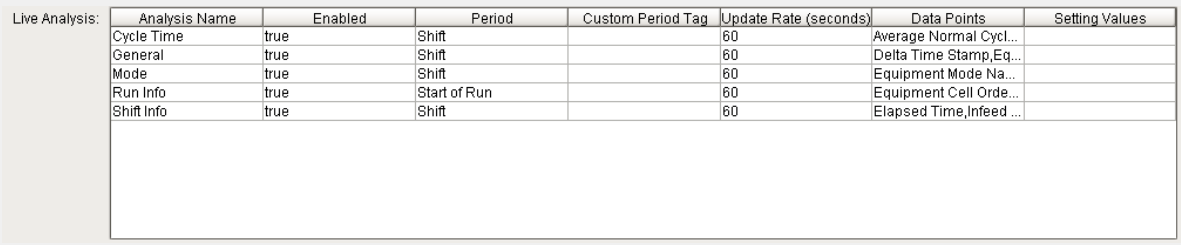
> Live Analysis provides a flexible way of customizing your application to provide a set of real-time tag values that can be accessed from the Ignition designer and used in your application to provide real-time production monitoring. Live Analysis is configured in the OEE 2.0 Downtime tab of the Production Model Designer for the Line, Cell Group and Cell

production items. When a Live Analysis is created, a corresponding set of tags is created in the MES Tag Provider that provide the real-time status of those datapoints based upon the Period defined for the Live Analysis. You can create multiple Live Analysis and use those tags to drive HMI displays.

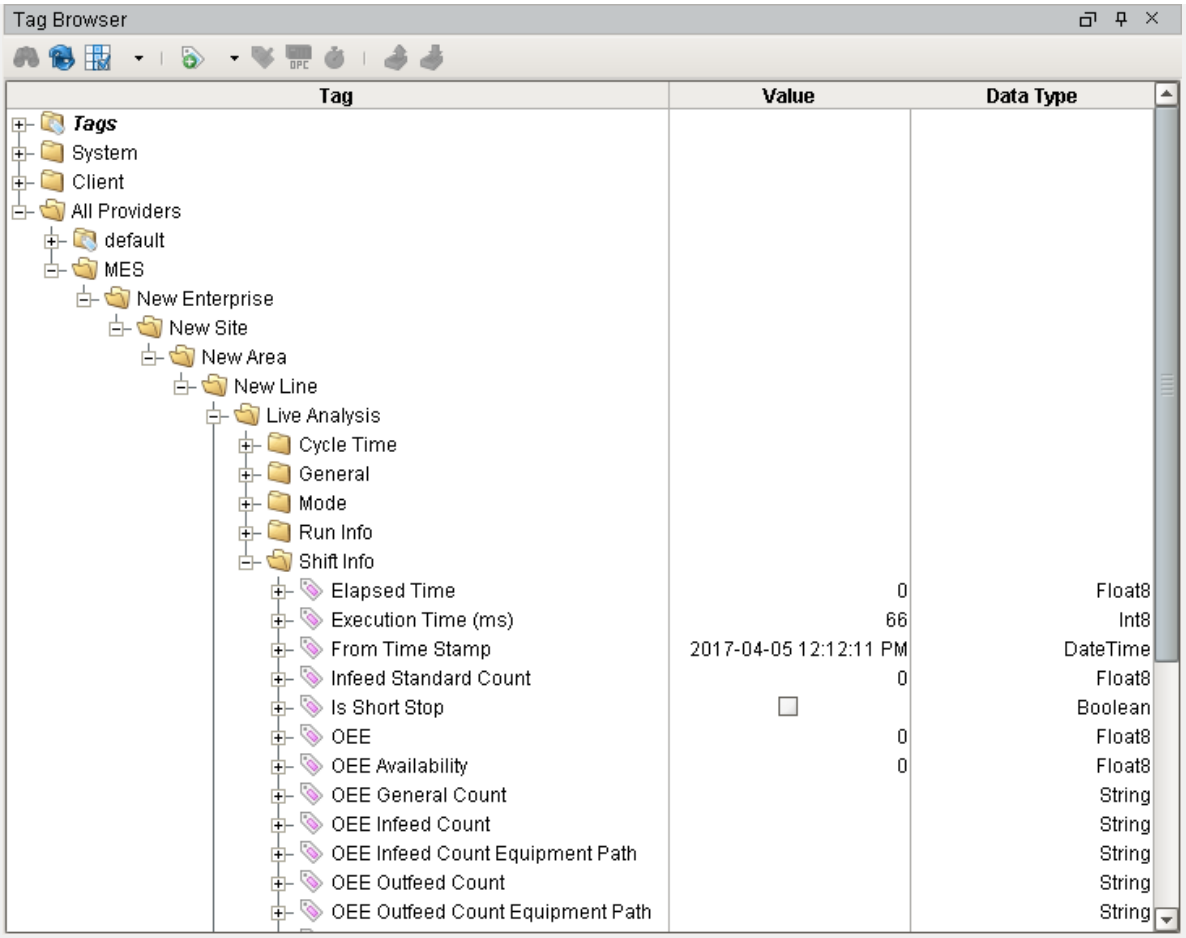
To create a new Live Analysis:

- Right click on the Live Analysis panel on the OEE 2.0 Downtime Tab in the Production Model Designer.
- Provide a Name
- Select the Period that the Live Analysis datapoints will return a value for. Valid options are Shift, Day (Midnight), Day (Production), Start of Run, Top of Hour, Custom Period Tag
- Select the frequency for how often the tag values will be updated. Default value is 60 seconds. Minimum value is 10 seconds
- Select the desired Data Points
- Add any further Settings Values required

You cannot select all Data Points in one Live Analysis. The maximum length string for Data Points is 1024 characters



Live Analysis Settings Panel in the OEE 2.0 Downtime Tab



MES Tag Provider Live Analysis Tags

Live Analysis Settings

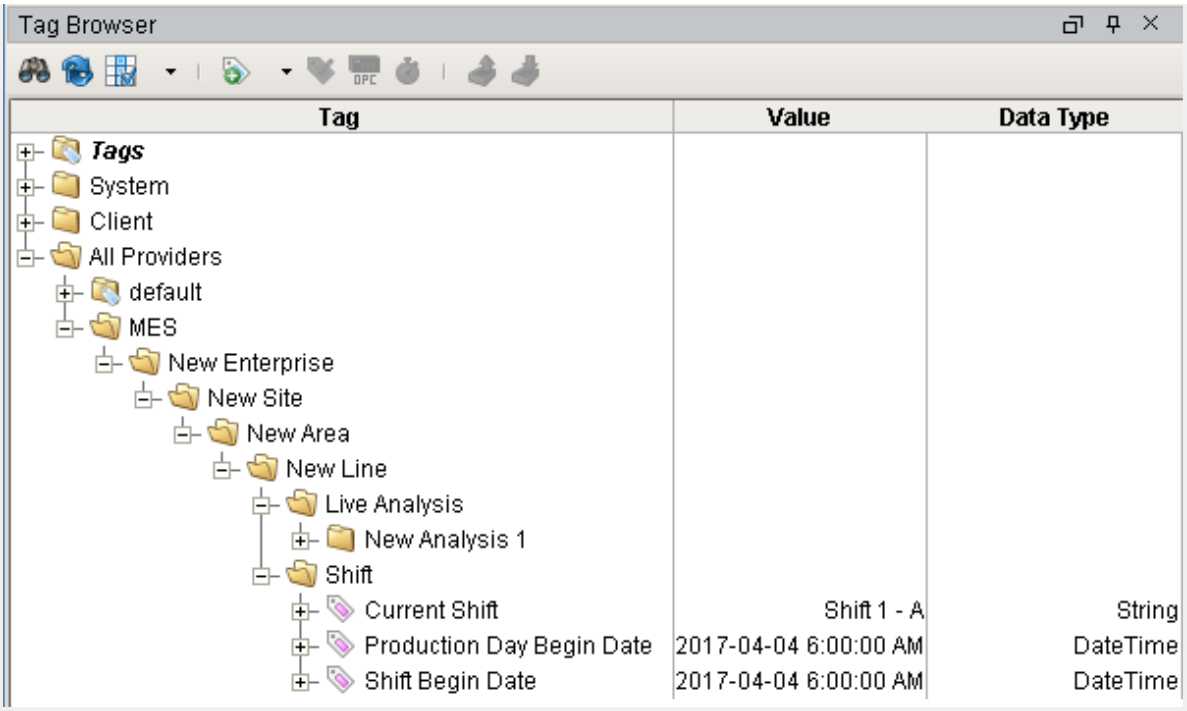
Setting	Description
Analysis Name	The name for the live analysis.
Enabled	The live analysis can be enabled or disabled with this setting.
Period	The duration of analysis can be set by **Shift**, **Day (midnight)**, **Day (production)**, **Start of Run**, **Top of Hour** or **Custom Period Tag**.
Custom Period Tag	A tag can be assigned to define the start datetime for a custom period. The end time will be the current time. It takes value in the date time data type. Example for a valid value for the custom period tag is: **`2017/04/04 14:00:00`**

Update Rate	The rate in seconds by which the live analysis is updated. The minimum update rate is 60 seconds.
Data Points	Data points allows you to pick and choose the values you wish to access through tags. See the table below for the listing of available data points.

Shift Data Points

When creating a Live Analysis, the following shift data points will be automatically created.

Data Point	Data Type	Description
Current Shift	String	The currently running shift as defined in the Ignition Schedule Management component or passed from the Shift Tag Collector path
Production Day Begin Date	DateTime	Production start time
Shift Begin Date	DateTime	Start time of the current shift



Analysis Data Points and Settings are used by Live Analysis, the MES Analysis Selector and MES Analysis Controller components, the MES Analysis Data Source for reporting and the MES Analysis Settings object.

Equipment Data Points

Data Point	Data Type	Description
Equipment		
Equipment Cell Order	Int4	Integer value that determines the cell order of the equipment within the line. Is set to *null* for the line. Is set to 0 for first cell within each cell group
Equipment Name	String	Name of the equipment as defined in the production model
Equipment Note	String	Any note that has been recorded for this piece of equipment through the Note tag collector path in the Production model will be exposed here.

Data Point	Data Type	Description
Equipment Operation Begin	DateTime	Start Date time of the currently running operation on this equipment
Equipment Operation Sequence	Int4	The ordinal number (integer) of operation
Equipment Path	String	Production model path for this equipment
Equipment Type	String	Can be Line, Cell Group or Cell
Execution Time (ms)	Int8	Time taken to execute and update the Live Analysis. Used mainly for performance debugging
From Time Stamp	DateTime	Start Date Time of current data point results
Infeed Units	String	See Infeed Units for more details
Is Key Cell	Boolean	See Key Reason for more details
Operation UUID	String	Unique Identifier for currently running operation
Outfeed Units	String	See Outfeed Units for more details
Product Code	String	Product code currently being processed on this equipment
Rate Period	String	See Rate Period for more details

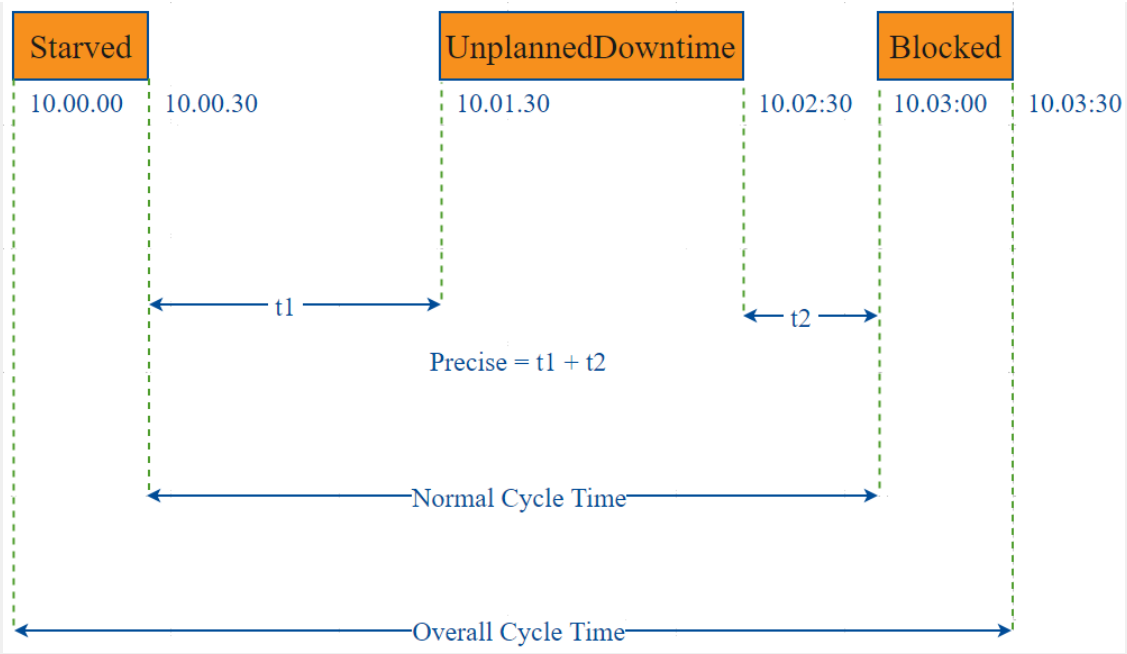
Data Point	Data Type	Description
Reject Units	String	See Reject Units for more details
To Time Stamp	DateTime	End Date Time of current data point results
Work Order	String	Work order currently being processed on this equipment

Equipment Count Data Points

Data Point	Data Type	Description
Equipment\Count		*Any defined counters for the production item will also appear in this folder
Equipment Infeed Scale	Float8	See Infeed Count Scale for more details
Equipment Package Count	Float8	See Package Count for more details
Equipment Reject Scale	Float8	See Reject Count Scale for more details
Outfeed-Material Out	String	Value of the default MES Counter used for OEE outfeed count

Equipment Cycle Time Data Points

The Cycle Time data points provides a number of metrics that can be used to measure the amount of time required to produce one piece. It is often used to gain an understanding of variations in production. Live Analysis provides Target, Normal, Overall and Precise Cycle Time metrics.



Data Point	Data Type	Description
Equipment\Cycle Time		
Relative Cycle Count	String	Relative Cycle Count is how many occurred for the compare by.
Target Cycle Time	Float8	Also known as Takt time, it is how often a piece must be produced to meet customer demand. It is often used to pace a production line, and it is a calculated number in seconds.
Total Cycle Count	String	Total Cycle Count is accumulative, it is sum total of all the cycle count.
Equipment\Cycle Time\Normal		**Normal Cycle Time is the actual cycle ignoring the equipment states like starved, blocked, etc.**
Average Normal Cycle Time	Float8	Average Normal cycle time in seconds for the time period selected
	Float8	

Data Point	Data Type	Description
Max Normal Cycle Time		Max Normal cycle time in seconds for the time period selected
Min Normal Cycle Time	Float8	Min Normal cycle time in seconds for the time period selected
Normal Cycle Time	Float8	Normal Cycle Time in seconds is the actual cycle ignoring the equipment states like starved, blocked, etc.
Equipment\Cycle Time\Overall		**Overall Cycle Time is the cycle including states like downtime, starved, blocked, etc.**
Average Overall Cycle Time	Float8	Average Overall cycle time in seconds for the time period selected
Max Overall Cycle Time	Float8	Max Overall cycle time in seconds for the time period selected
Min Overall Cycle Time	Float8	Min Overall cycle time in seconds for the time period selected
Overall Cycle Time	Float8	Overall Cycle Time in seconds is the cycle including states like downtime, starved, blocked, etc.
Equipment\Cycle Time\Precise		**Precise Cycle Time is the cycle time ignoring all the equipment states**
Average Precise Cycle Time	Float8	Average Precise cycle time in seconds for the time period selected
Max Precise Cycle Time	Float8	Max Precise cycle time in seconds for the time period selected
Min Precise Cycle Time	Float8	Min Precise cycle time in seconds for the time period selected

Data Point	Data Type	Description
Precise Cycle Time	Float8	Precise cycle time in seconds excluding states like planned downtime, unplanned downtime, starved and blocked.

Line Data Points

The Line Data points returns data for the line regardless of the Equipment the Live Analysis has been set up for.

Data Point	Data Type	Description
Line /Downtime		
Line Downtime Equipment Name	String	Name of the equipment that is responsible for causing line downtime
Line Downtime Equipment Path	String	Production model equipment path for equipment that is responsible for causing line downtime
Line Downtime Event Sequence	Int4	Every downtime event on the line is provided with an incrementing sequence number
Line Downtime Note	String	Note entered at the Line level
	Int4	Number of downtime events for the selected period.

Data Point	Data Type	Description
Line Downtime Occurrence Count		
Line Downtime Reason	String	The line or cell group (sub line) downtime reason. 1. When the line is down the Line Downtime Reason is the same as the Line State Name. 2. When the line is up the Line Downtime Reason is blank.
Line Downtime Reason Path	String	The full reason name for line or cell group (sub line) downtime reason. Line State name including State Class i.e. *Default/Cell Faulted*
Line Downtime Reason Split	Boolean	The line downtime reason split indicator. True is current downtime event has been split into multiple downtime events
Line Downtime State Time Stamp	DateTime	The time stamp for the equipment state change of the cell group (sub line) or cell that caused the line down time even.
Line /Meantime		
Line MTBF	Float8	The calculated Meantime (minutes) Between Failure for the selected period. Refer to Setting Up Equipment States - Meantime Metrics for more details.

Data Point	Data Type	Description
Line Meantime Metrics Enabled	Boolean	Returns if Meantime metrics have been enabled for this equipment.
Line /Schedule		
Line Schedule Available	Boolean	True if this operation was scheduled
Line Schedule Available Time	Float8	Time in minutes for available production time adjusted for line schedule availability and mode.
Line Standard Count	String	Amount of product that should have been produced based on the line schedule available time and line standard rate
Line Standard Count Variance	String	Variance between standard count and actual count
Line Target Count	String	Amount of product that should have been produced based on the line schedule available time and line schedule rate
Line Target Count Variance	String	Variance between line scheduled count and line OEE outfeed count.
Schedule Rate	Float8	See Schedule Rate for more details
Line/State		

Line State Duration	Float8	The line or cell group (sub line) downtime event duration in minutes.
Line State Event Begin	DateTime	The line or cell group (sub line) downtime event begin date time.
Line State Event End	DateTime	The line or cell group (sub line) downtime event end date time.
Line State Event Sequence	Int4	The equipment state event sequence number.
Line State Name	String	The line or cell group (sub line) state. 1. When the line is down the Line Downtime Reason is the same as the state name. 2. When the line is up the Line Downtime Reason is blank.
Line State Override Scope	String	The state override scope for a line or cell group (sub line). See Setting Up Equipment - Override Scope for more details
Line State Override Type	String	The state override type for a line or cell group (sub line). See Setting Up Equipment - Override for more details
Line State Type	String	The line or cell group (sub line) state type. See Setting Up Equipment - State Type for more details
Line State Value	Int4	

Data Point	Data Type	Description
		The line or cell group (sub line) downtime state code. See Setting Up Equipment - State Code for more details

Equipment Mode & State Data Points

Data Point	Data Type	Description
Equipment /Mode		
Equipment Mode Name	String	Name of the current mode. See Setting Up Equipment Modes for more details
Equipment Mode Type	String	Name of the current mode type. See Setting Up Equipment Modes for more details
Equipment Mode Value	Int4	Name of the current mode. See Setting Up Equipment Modes for more details
Mode Begin Time	DateTime	Start time of the current mode
Mode Duration	Float8	Duration of the current mode in minutes
Mode End Time	DateTime	End time of the current mode
OEE Enabled	Boolean	See Setting Up Equipment Modes - OEE Enabled for more details
Production Counts Enabled	Boolean	See Setting Up Equipment Modes - OEE Enabled for more details

Data Point	Data Type	Description
Equipment /State		
Equipment Original State Value	Int4	The original value of equipment state tag collector before it is updated by using MES Value Editor component or scripting
Equipment State Name	String	Current state name
Equipment State Path	String	Production model equipment path for equipment that is responsible for causing line downtime
Equipment State Split	Boolean	True is current downtime event has been split into multiple downtime events
Equipment State Type	String	See Setting Up Equipment - State Type for more details
State Begin Time	DateTime	Start time of the current state
State Duration	Float8	Duration of current state in minutes
State End Time	DateTime	End time of the current state

Equipment Meantime Data Points

Data Point	Data Types	Description
Equipment/Meantime		
Equipment MTBF	Float8	

Data Point	Data Types	Description
		The Mean Time (minutes) Between Failure for the selected period
Equipment Meantime Metrics Enabled	Boolean	True if Equipment Meantime Metrics are enabled for the current equipment state

Equipment General Data Points

Data Point	Data Types	Description
Equipment /General		
Delta Time Stamp	Float8	Time gap (minutes) between the rows of data.
From Time Stamp	DateTime	Start time (minutes) of the current period
Shift	String	Name of the current shift as set by the Ignition Schedule Management component and defined for the current line or by the value passed in the equipment shift tag collector
Shift Day Text	String	Name of the current day
Shift Day of Month	Int4	Int value of the current month
Shift Day of Week	Int4	Int value of the current day of the week
Shift Day of Year	Int4	Int value of the current day of the year

Data Point	Data Types	Description
Shift ISO Week of Year	Int4	Int value of the ISO week of the year
Shift Month Text	String	Name of the current month
Shift Month of Year	Int4	Int value of the current month of the year
Shift Start Date	DateTime	Start time of the current shift
Shift Week of Month	Int4	Int value of the current week of the month
Shift Week of Year	Int4	Int value of the current week of the year
Shift Year	Int4	Int value of the current year
To Time Stamp	DateTime	Endtime (minutes) of the current period

Equipment OEE Data Points

Data Point	Data Type	Description
Equipment/OEE		
Elapsed Time	Float8	Elapsed Time of current operation
OEE	Float8	OEE value for selected period

Data Point	Data Type	Description
OEE General Count	Long	Any count value other than infeed, outfeed, reject and waste value for the selected time period
OEE Infeed Count	Long	Equipment infeed count value for the selected period
OEE Infeed Count Equipment Path	String	Infeed count tag collector path
OEE Outfeed Count	Long	Equipment outfeed count value for the selected period
OEE Outfeed Count Equipment Path	String	Outfeed count tag collector path
OEE Reject Count	Long	Equipment reject count value for the selected period
Planned Downtime	Float8	Planned Downtime duration (Double) for selected period
Runtime	Float8	Planned Downtime duration (Double) for selected period
Short Stop Time	Float8	Short stop duration (Double) for selected period
Standard Rate	Float8	See standard rate for more details
Target Changeover Time	Float8	Amount of time in minutes set for Target Changeover. See Changeover Duration for more details
Unplanned Downtime	Float8	Unplanned Downtime duration (Double) for selected period

Data Point	Data Type	Description
Equipment/OEE /Availability		
Is Short Stop	Boolean	True if current equipment state is consider a shortstop. See Short Stop Threshold for more details
OEE Availability	Float8	OEE Availability value for selected period
Equipment/OEE /Performance		
OEE Performance	Float8	OEE Performance value for selected period
Infeed Standard Count	Float8	Calculated expected infeed based on standard rate
Equipment/OEE /Quality		
OEE Quality	Float8	OEE Quality value for selected period

Setting Values

The analysis results that are returned can be modified through the use of settings. Setting values provide a number of keywords as listed below.

Format for entering the keywords is k*eyword1=True, keyword2=100.0, keyword3=10.*

> ⚠ Settings like Enable Totalized Mode, Include Future, Last Values and Rollup Time span is meant for analysis selector and not for live analysis

Setting	Description	Use	Example
Date Format		All	

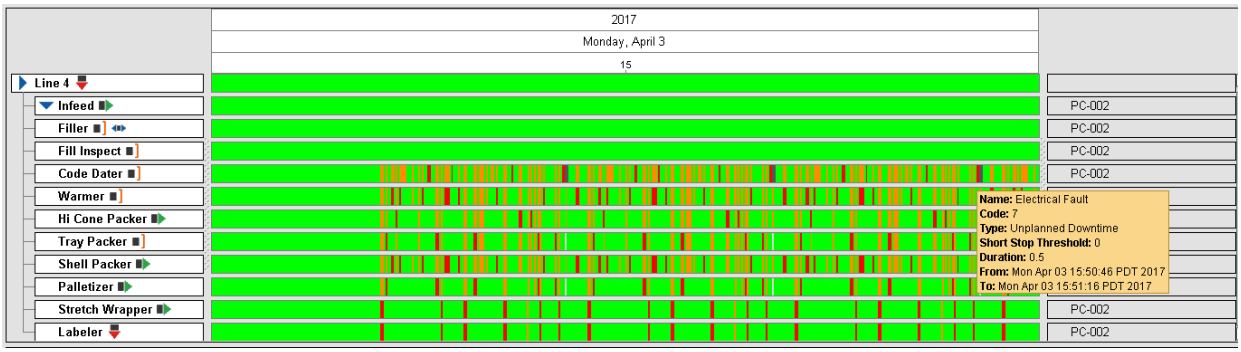
Setting	Description	Use	Example
	Date format fields can be customized with this setting e.g. 'YYYY/MM/dd hh:mm:ss a'		Date Format = `2017/04/12 19:45:30`
Enable Totalized Mode	This setting accumulates the count. Useful for charts where you wish to display the accumulated production count over time	Not valid for Live Analysis	Enable Totalized Mode = True
Include Future	Allows for count values to be calculated in the future. Useful for charts where you want to display target counts for future runs	Not valid for Live Analysis	Include Future = True
Last Values	Only the latest values are shown.	Not valid for Live Analysis	Last Values = True
OEE Availability Cap	The maximum value calculated can be capped with this setting	All	OEE Availability Cap = 100.0
OEE Performance Cap	The maximum value calculated can be capped with this setting	All	OEE Performance Cap = 100.0
OEE Quality Cap	The maximum value calculated can be capped with this setting	All	OEE Quality Cap = 100.0
Rollup Time Span	If the time (seconds) between downtime events is less than the rollup time and it is the same equipment and reason, then it will rollup the event into one row in the results and will increase the occurrence count.	Not valid for Live Analysis	Rollup Time Span = 30

Setting	Description	Use	Example
Row Limit	The analysis can be limited to a certain number of rows.	All	Row Limit = 10

Production Run Components

OEE Time Chart

The OEE Time Chart component provides a visual of exactly what is happening on your production line. It can be configured to show modes as well as states, and to also show line schedule information. The time range to show data for is configurable through the start date and end date properties. You will need to provide an equipment path to the line.



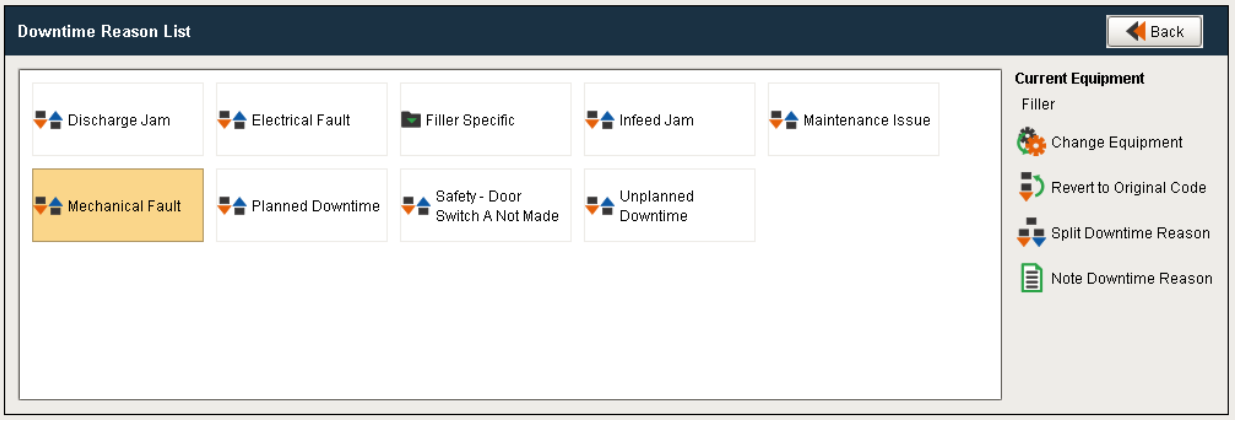
Current equipment status information is displayed on the left hand side and is based on the equipment state type. On the right hand side, extension functions provide you with a method to customize this component by adding any type of equipment data you wish to show.

For more information please refer to the OEE Time Chart component help in the reference section.

OEE Downtime Table

The OEE Downtime Table component displays all events that caused the production line to go down. This component can be used to display line downtime events for the currently running process as well as for production runs in the past. It also provides a method for modifying the cause of a line downtime event and to add notes about the cause.

Begin	End	Cell	Reason	Downtime Minutes	Note
4/5/2017 5:36 PM	4/5/2017 5:37 PM	Stretch Wrapper	Infeed Jam	1	
4/5/2017 5:35 PM	4/5/2017 5:36 PM	Filler	Infeed Jam	0.5	
4/5/2017 5:35 PM	4/5/2017 5:35 PM	Fill Inspect	Unplanned Downtime	0.47	Inspection head out of alignment
4/5/2017 5:34 PM	4/5/2017 5:35 PM	Code Dater	Electrical Fault	0.5	
4/5/2017 5:33 PM	4/5/2017 5:34 PM	Filler	Mechanical Fault	1.5	
4/5/2017 5:33 PM	4/5/2017 5:33 PM	Stretch Wrapper	Infeed Jam	0.5	Out of wrap
4/5/2017 5:32 PM	4/5/2017 5:33 PM	Tray Packer	Electrical Fault	1	
4/5/2017 5:31 PM	4/5/2017 5:31 PM	Code Dater	Mechanical Fault	1	
4/5/2017 5:30 PM	4/5/2017 5:30 PM	Filler	Safety - Door Switch A Not Made	0.5	
4/5/2017 5:29 PM	4/5/2017 5:29 PM	Fill Inspect	Unplanned Downtime	1	
4/5/2017 5:27 PM	4/5/2017 5:28 PM	Warmer	Safety - Door Switch A Not Made	1.47	
4/5/2017 5:27 PM	4/5/2017 5:27 PM	Filler	Safety - Door Switch A Not Made	0.5	
4/5/2017 5:26 PM	4/5/2017 5:27 PM	Hi Cone Packer	Electrical Fault	0.5	
4/5/2017 5:26 PM	4/5/2017 5:26 PM	Filler	Mechanical Fault	1	
4/5/2017 5:25 PM	4/5/2017 5:25 PM	Tray Packer	Electrical Fault	0.5	
4/5/2017 5:24 PM	4/5/2017 5:25 PM	Stretch Wrapper	Discharge Jam	0.97	



For more information please refer to the OEE Downtime Table component help in the reference section.

Analysis and Reporting

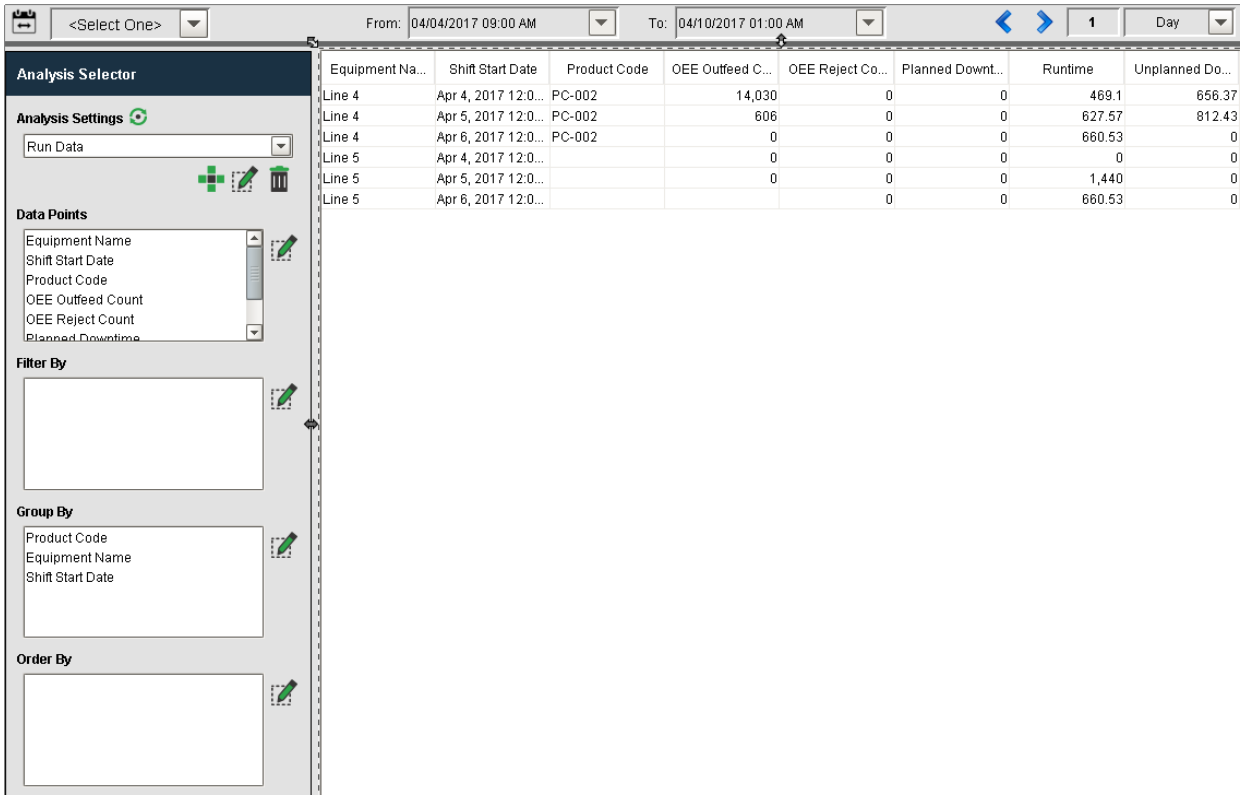
Analysis of production data is provided through a set of analysis components and scripting functions. The analysis engine handles the aggregation and calculation of raw production data stored in the SQL tables and turns it into KPI's and metrics that can be used in charts, dashboards, reports, and shared with other information systems.

MES Analysis Selector

At the heart of analysis is the Analysis Selector component that allows you to create, manage and disseminate production information throughout your organization.

Refer to MES Analysis Selector for more details.

> Existing reports will appear in the drop down menu of analysis settings.

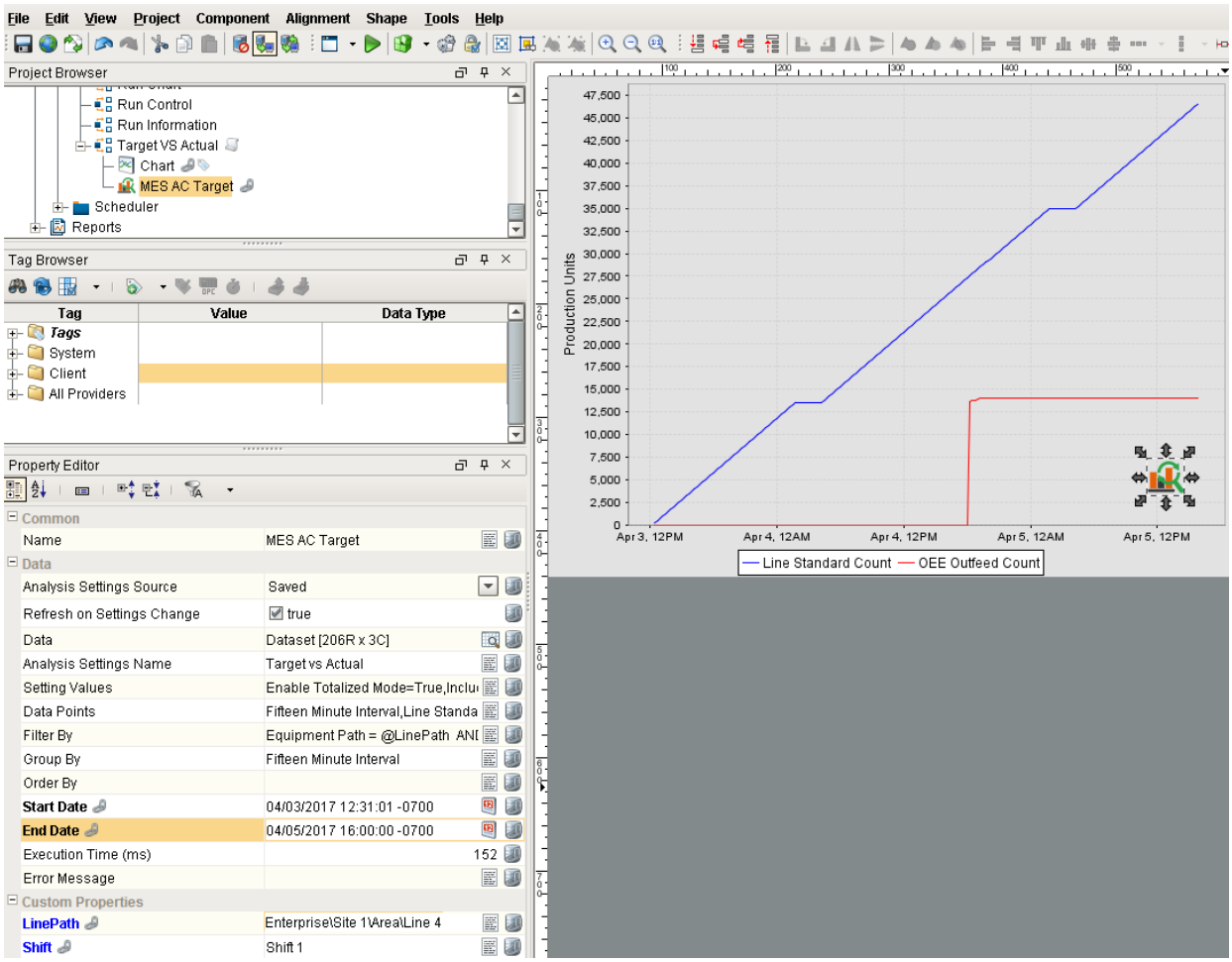


MES Analysis Controller

The Analysis Controller is an invisible component that is visible in the designer, but doesn't appear in the client. This component allows you to pull analysis data in the same way as the Analysis Selector component to return a dataset that can be used to populate charts, tables, reports and any ignition component that accepts a dataset as a property.

The analysis controller can be pointed to a saved stored analysis created using the Analysis Selector component, or it can be configured to use property values as set on the component itself.

Refer to MES Analysis Controller for more details.



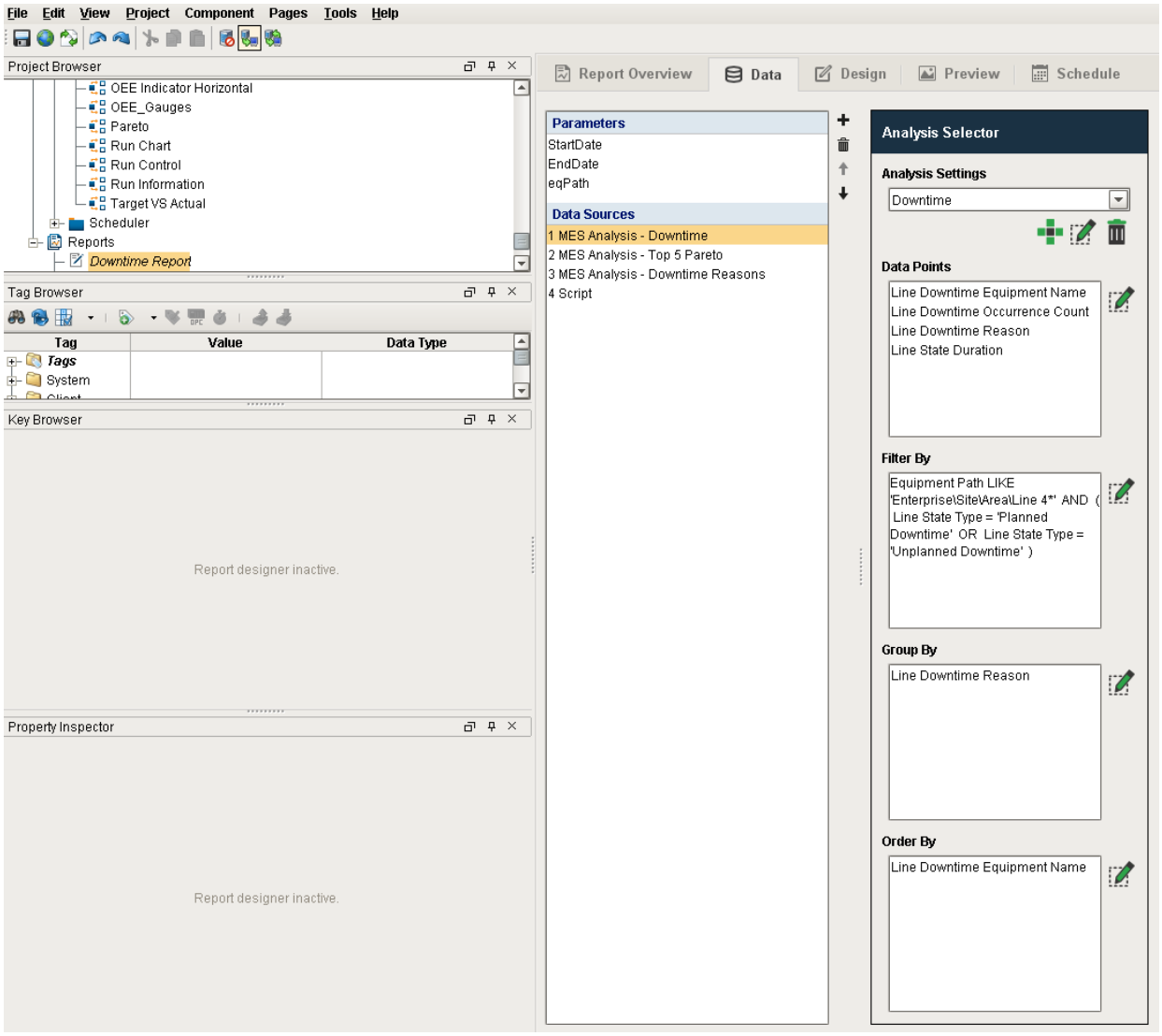
Scripting Functions

All the capabilities of the MES Analysis Selector and MES Analysis Controller components are available through scripting functions. This allows for the dissemination of analysis data based on events such as shift change, or end of production and can be used to update other Information systems. The following script functions are available for analysis

Creating Production Reports using the MES Analysis Data Source

Data returned by the analysis engine can easily be passed to the Ignition Report Designer to create Production Reports. In the Report Designer, you can add a data source of type **MES Analysis**. This brings up the Analysis Selector component where you can select a saved analysis or create a new analysis to be used by the report.

Refer to the Ignition help on the Reporting Module for more details.



8.2 Track & Trace

New to Track & Trace?

Download and test drive this most powerful MES solution available anywhere!

Download and Install Module

A Simple Workflow

Step 1. Database Connection

Step 2. Configuring MES Databases

Step 3. Installing the Production Simulator

Step 4. Production Model Configuration

Track and Trace Module in a nutshell

Click on the topic you would like to learn more about ...

- Components
- Scripting
- Objects
- Binding Functions

Take the MES course to boost your knowledge

Product Data Sheet

To see the Product Data Sheet,

click on **Track and Trace Module**

Click here to see Knowledge base articles of Track and Trace.

Info

Sepasoft Track and Trace module uses system.mes functions for scripting.

Read this section about licensing...

Licensing and Activation

8.2.1 Track & Trace Overview

Quick Product Feature Overview

This paperless and fully integrated solution allows you to track finished goods from their raw materials to their finished state, access genealogy data and set up a centralized operator interface for all MES information. Along with tracking products through the manufacturing process, it can also pull production information together from various sources such as OEE, SPC, Recipe, Process Historian and more. Data can be queried from systems outside of Ignition, such as an ERP, CMMS or a warehouse management system.

All this product information can be pulled together through a unique and easy-to-navigate Trace Graph display that makes viewing the finished goods process data quick and easy. This information is organized chronologically and displayed in one place, so it's simple to find the exact information you need. For example, view trends within Ignition, query external systems, and see correlations between efficiency and quality.

The Track and Trace Module allows your enterprise to transition away from recording traceability information on paper. Instead, you can keep it alongside the rest of your system data on a unified platform where it’s easy to analyze and retrieve. The Track and Trace Module can help meet the increasing regulatory demand for traceability information, by tracking products through the supply chain and making it possible to quickly access traceability information requested by government authorities. This can help you avoid hefty fines.

Seamlessly integrated with the Ignition platform, the Track and Trace Module ensures accurate product information and adds meaningful context to time-series data. The module is built for Ignition and shares the same advantages, such as cross-platform compatibility, unlimited free clients, robust out-of-the-box SQL database support, and fast installation.

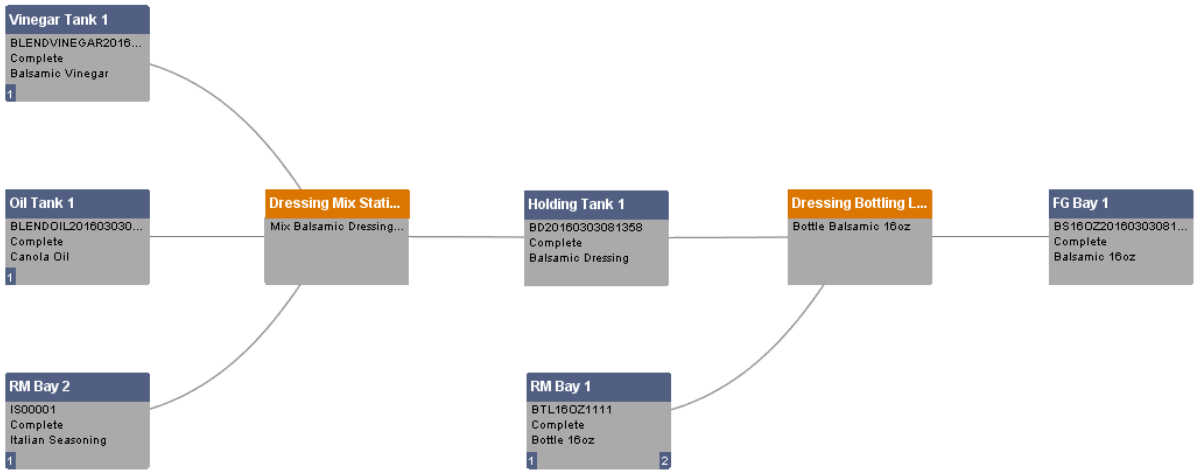
The Track and Trace module is built on the ISA-95 standard, the international model for integrating enterprise and control systems. By leveraging the full power of Ignition, the Track and Trace module is unmatched by any other traceability application on the market.

Overview

Knowing where product is and has been in a production facility can be very valuable. In the typical production environment, time series data is collected by the SCADA or HMI systems. Giving that data context to the specific product being produced, or other criteria, can provide a picture into your process that is very valuable when diagnosing quality or other issues. It is also valuable for narrowing down recalls and ensuring regulatory compliance.

By adding the Track & Trace Module, your system can have the capability to look up where any product has been in its manufacturing process, and where it is now. This paperless and fully integrated solution allows you to do the following:

- Track products from the raw materials to the finished state, including consumables and byproducts
- Access genealogy data
- Set up unified operator interfaces including SCADA, HMI, other MES, and more
- Serialization of items and sub-assemblies
- Real time inventory management



The Track & Trace Module is built on the ISA 95 Standard. For more information on ISA-95, you can go to ISA-95 Overview

Automated Traceability Software

Track and Trace is the process by which manufacturers obtain and record highly important information about where and how products are made. Track and Trace software automates this process and has become a modern necessity for manufacturers as their industry faces increasing economic and regulatory challenges. Track and trace software fits into the Manufacturing Execution Systems (MES) / Manufacturing Operations Management (MOM) layer that resides between the Enterprise Resource Planning (ERP) layer and the plant floor.

Track and trace module records the start and end of each production run in real-time and monitors the status of the materials being consumed. This information is desirable to manufacturing plants for many reasons. One benefit for managers is evaluating the manufacturing process. If operators are adding materials incorrectly, or adding the wrong materials, controls can be put in place to ensure operators work in the desired manner. Another benefit is the accessibility of critical data during a recall.

For every product you will be able to track all the consumed parts, the suppliers that supplied the parts, the people that worked on the product and when they worked on it, the equipment that was used to manufacture the part, Lot numbers, Serial Numbers, measurement data that was acquired and additional factors such as rework etc.

The Track and Trace module extends Ignition to manage and track production and then provide trace results. It is ideal for quickly implementing track and trace systems without the need to design database schemas because it is handled by the module. New Ignition components are also included that eliminate the need to build custom screens with entry boxes for each of the values accepted by the user or barcode scanner. Also included, is a powerful visual management component to define material, personnel, equipment, production tasks, routes, etc. Then trace results are visually analyzed using the trace graph component.

Knowing where product is and has been in a production facility can be very valuable. In the typical production environment, time series data is collected by the SCADA or HMI systems. Giving that data context to the specific product being produced, or other criteria, can provide the picture into your process that is very valuable when diagnosing quality or other issues. It is also very useful for narrowing down recalls and ensuring regulatory compliance.

Tracking Product

In the real-world production environment, tracking product is not as easy as it sounds. It impacts production staff and can hinder their efficiency. In some cases tracking information may be provided by another system and in other cases it might require user input. It is preferable that the user input is done real-time as opposed to the operator writing on paper and a data entry person entering it into the system. In general, an effective system will have a minimal impact of operations staff.

Tracing Product

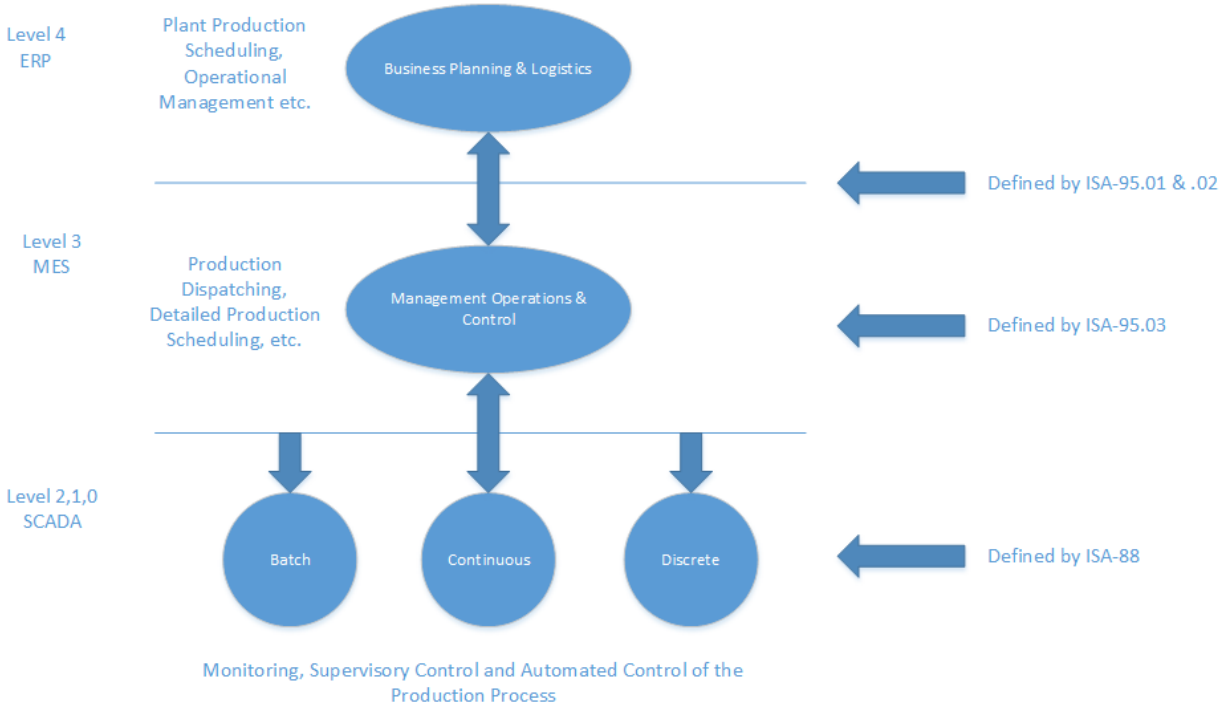
This is the analysis and reporting of the details that went into making or processing a product. The Sepasoft MES system includes linking other data such as data from the historian, OEE, SPC and recipe modules or even data from external systems.

8.2.2 Implementing Track & Trace

The Track & Trace module provides several distinct features...

- Production Control Tracking
- Traceability
- Onhand Lot Inventory

The project requirements, existing systems, and architecture will all determine how Track & Trace is implemented. It may be that the ERP will manage raw material assignment and high levee production scheduling, in which case Track & Trace can simply be the vehicle used to provide real-time updates of material transfers on the plant floor to the ERP system or inventory management system. The ERP system may also contain the rules that govern where materials can be received or stored, in which case Track & Trace can interface between the user and the ERP to provide the necessary production control. Remember that even MESA states that the lines between the ERP (level 4) and MES (level 3) layers are somewhat blurred. The trick in an MES implementation is to understand the capabilities of all systems and determine where it makes sense to implement functionality and the appropriate interfaces.



Production Control

Production Control can enforce where certain operations occur, who can perform the operations, what materials may be received, where they can be stored and how they are stored (random lot, single lot or blended lot). The Track & Trace module can be configured to tightly control the transfer of materials within the manufacturing environment. This level of control is implemented by the Operations Definition and the structure used to define Material Groups. Production Control can only be implemented via a User Interface, where an operator will interact with a screen to select an operation, the incoming material and where it will be stored. When the Track & Trace system obtains its information through tag values (i.e. pumps and valves turning on), a much looser implementation must be employed to correctly track the transfer of materials from one location to another.

Traceability

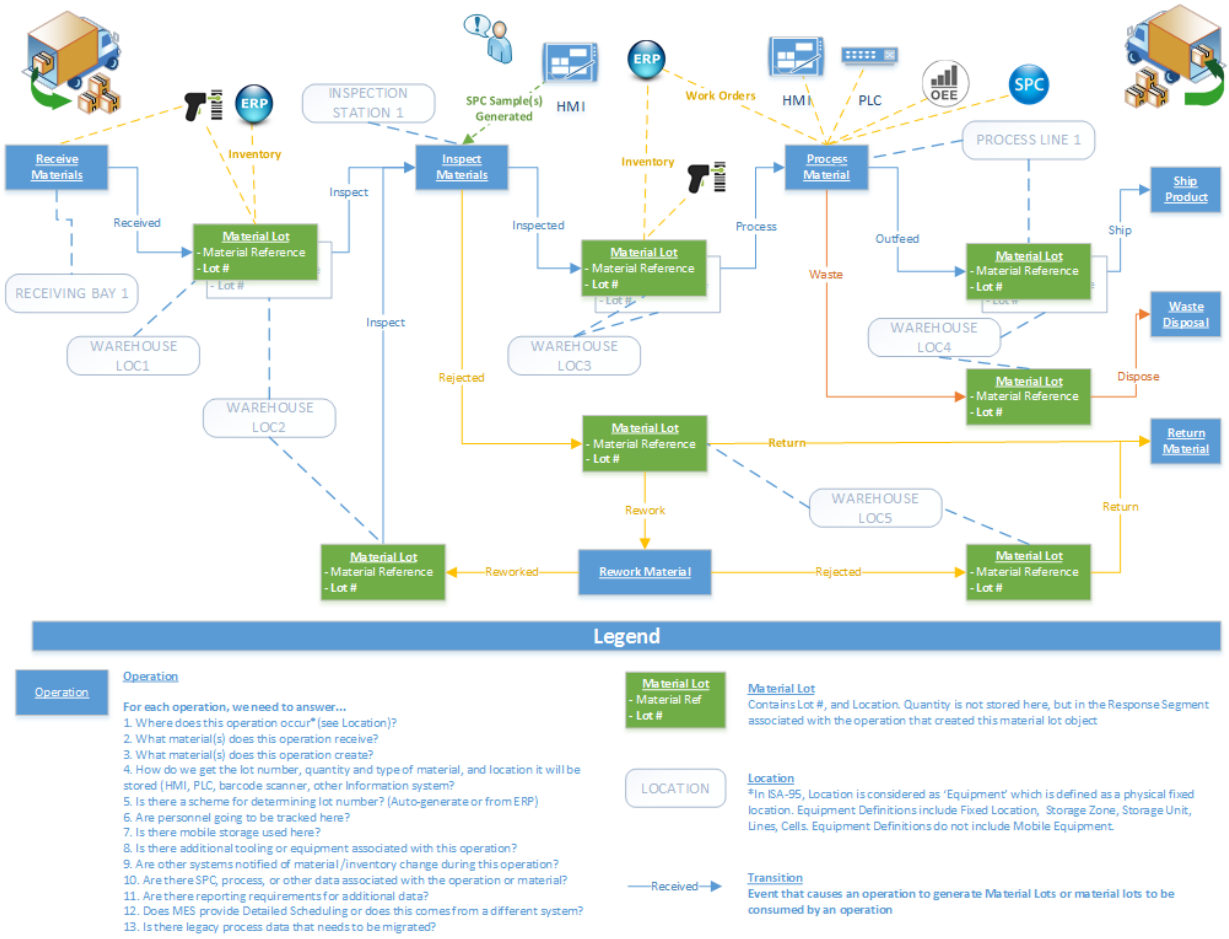
A standard requirement of a Track & Trace system is the ability to trace what materials were used to make a finished product and what processes were employed during the manufacturing process. Understanding the details of the type of end user analysis required will drive how granular the material flow tracking will be and what data (custom properties) need to be stored.

Onhand Inventory

Onhand Inventory of what materials are available and where they are may be a system requirement as well as historical inventory. The Track & Trace module can provide this if required. This requirement will drive how the manufacturing process and storage units are configured.

Modeling the Material Flow

Understanding the material flow through the manufacturing process, from Raw Materials In, to Finished Goods, is the first step in understanding how Track & Trace should be implemented to model the business. In the act of mapping out the flow, touch points with other information systems such as ERP, Inventory Management Systems etc. will be defined, as well as understanding where the information (lots ids, quantities, etc.) will come from (bar code scanner, manual entry screen, PLC tags).



The drawing above shows an example of a Material Flow artifact that should be generated prior to any development work. You can download a visio version here to help kick off your material flow.

Track & Trace Configuration

There are a number of steps to configuring the Track & Trace Module.

1. Define Fixed Equipment (Production Lines and Cells) and Storage Locations (Storage Zones and Units).
    a. Fixed Equipment and Storage Locations are configured in the Production Model in the Designer. See Adding Fixed Equipment and Storage Locations for more details
2. Configure Fixed Equipment Lot Handling Modes
3. Create Equipment Classes
    a. Logical grouping of equipment configured in the MES Management screen
4. Create Supplemental Equipment
5. Create Material Classes and Definitions

    a. Logical grouping of materials configured in the MES Management screen

6. Create Personnel Classes and Definitions
    a. Logical grouping of operators etc., configured in the MES Management screen.
7. Create Operations, Process Segments and Routes
    a. Configured in the MES Management screen

The Online Tutorial provides step by step instruction on configuring equipment, materials and operations for an example Track & trace project.

Adding Fixed Equipment and Storage Locations

After a Material Flow has been created for the Track & Trace project, the fixed equipment that will be used for material storage and processing should be defined in the Production. The information below discusses the production model. Note, mobile equipment that can be moved around such as bins, containers, die sets etc. should not be defined in the production model as fixed equipment, but in the MES Management screen as Supplemental Equipment.

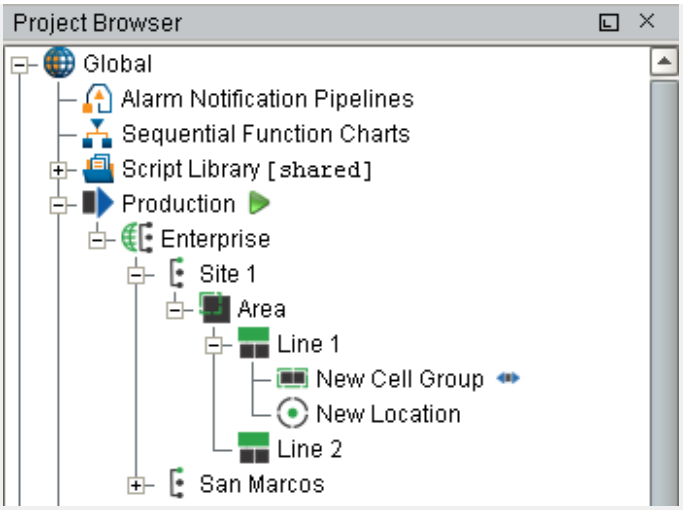
Help Manual - Production Model Overview

What Is The Production Model?

When any of the core MES modules (OEE, SPC, Recipe, T&T) are installed, the Production Model is added to the **Global** project resources in the Project Browser window of the Ignition Designer. The Production Model allows you to define your manufacturing process in a tree view form and provides an organized way to configure, control, and analyze your manufacturing activities. It provides the foundation on which the MES modules are built.

The Production Model is a hierarchy of Sites, Areas, Lines, Cell Groups, Cells, Locations, Storage Zones and Storage Units. Typically, Lines and Cells are used to represent machinery or equipment where a process occurs transforming raw materials into sub-assemblies or finished goods. Storage Zones and Storage Units are typically used to define where to get or store material.

Lines and Cells defined in the production model should be considered to be equipment that is bolted to the floor and has conduit running to it. Mobile equipment such as pallets, bins, dies used for pressing, etc. are not defined in the production model, but configured in the MES Management screen as Supplemental Equipment (Track & Trace only).



Below are the different types of Production Items that can be added to the production model.

Icon	Production Item	Description	Module
	Enterprise	The enterprise is the highest level of the production model and typically represents a manufacturing company. You can rename the Enterprise production item to your companies name. You can only have one Enterprise item in the Production Model.	All
	Site	A site is a fixed geographical production location that is part of an enterprise. Separating your enterprise into multiple production sites allows for comparing OEE, downtime and production information between them.	All
	Area	An area is a physical or logical grouping of production lines.	All
	Line	A line is a collection of one or more cells and/or cell groups that work together to perform a sequence of process steps. Typically, the product flows from one cell or cell group to the next in sequence until the product, or sub assembly, being produced is complete.	All

Icon	Production Item	Description	Module
		Understanding how Operations schedules or controls a production run will help in determining whether cells should be grouped into a line or be considered lines themselves.	
	Location	A location item is the place where a sample is collected. This can be placed under an area or a line.	SPC
	Cell Group	A cell group contains two or more cells. Typically, these cells occur at the same time in the sequence of the line instead of one after another, causing the cell group to act as a single sub process or step within the production.	All
	Cell	The cell is a single machine, sub process or step required in the manufacture of a product. The product may be a hard product such as used in packaging, adding liquid or powder, etc. Packaging machines are a common example, but a cell applies to processes also.	All
	Storage Zone	A storage area such as a warehouse.	T&T
	Storage Unit	A storage unit located inside of a storage zone. For example, you may have a warehouse with bay 1 to 5.	T&T

Configuring the Production Model

The production model is configured within the Ignition designer and is accessed by selecting the **Production** node under Global in the project browser. From here your enterprise, site, area(s), line(s) and line cell(s), line cell group(s), storage zone(s) and storage unit(s) can be added, renamed and deleted.

It is extremely important to understand production OPC values have an OPC item path that matches the layout of the production model and that renaming production items can cause Ignition tags associated with a production item to stop being updated.

Adding a New Production Item

To add a new Production item, right-click on the Production model and select the **New Production Item** > **New Production xxxx** menu item.

Renaming a Production Item

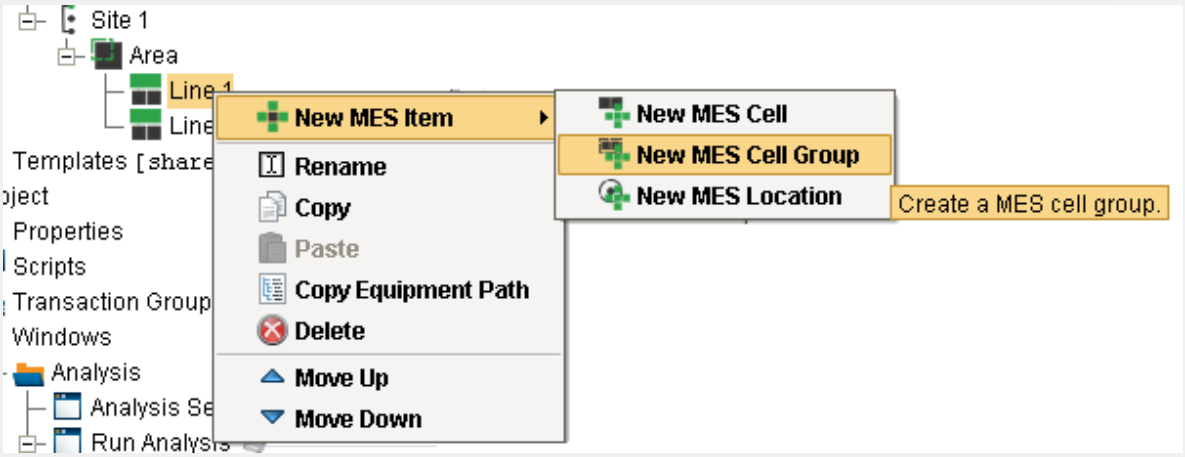
To rename a production item, right-click on it and select **Rename**, then enter the new name.

> Please note that when you rename a production item, it actually creates a new instance of a production item and disables the old production item. This is important to note as data captured against that production item will not be accessible to the newly renamed production item. Spend the time to get the Production Item named correctly at the beginning of the project.

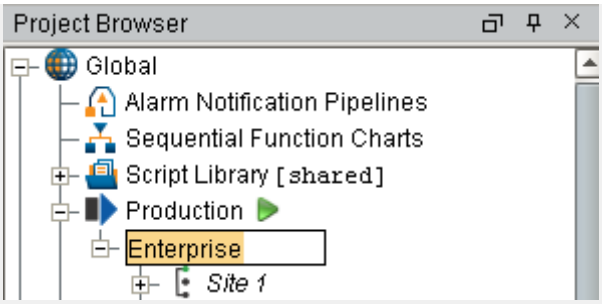
Deleting a Production Item

To remove an existing production item, right-click on the item and select the **Delete** menu item. A window will appear confirming that you permanently want to delete the production item.

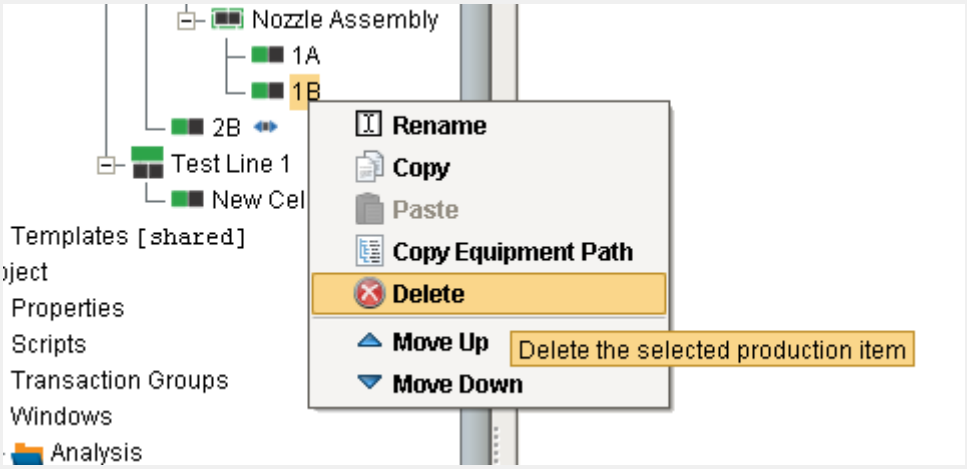
> Please note that any line(s), cell(s), cell group(s) and location(s) underneath the production item will also be permanently removed.



Adding a new Cell Group to the Production Model



Renaming the Enterprise



Delete a Cell

Copying a Production Item

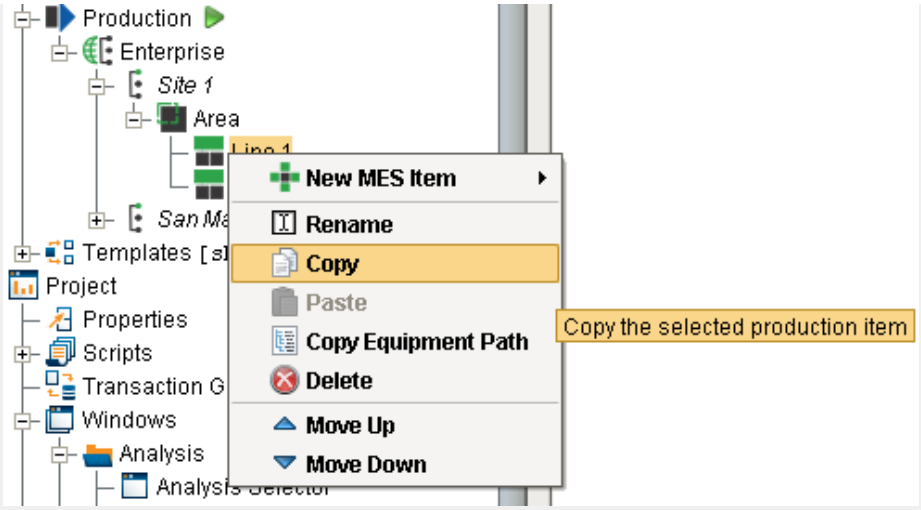
Right Click mouse button and select **Copy** on any production item to copy that production item.

Right Click mouse button and select **Paste** to make a copy of that production item in the production model.

If you are copying a line, select the line before copying it. When you paste it, select the area in which to the create a copy of that line.

Good Practice

It is recommended that you make a gateway backup prior to copying and pasting Production items. It is not recommended that you make changes to the production model on the production server without scheduling with Operations and having the system backed up.



Copying a Production Item

Production Item General Settings

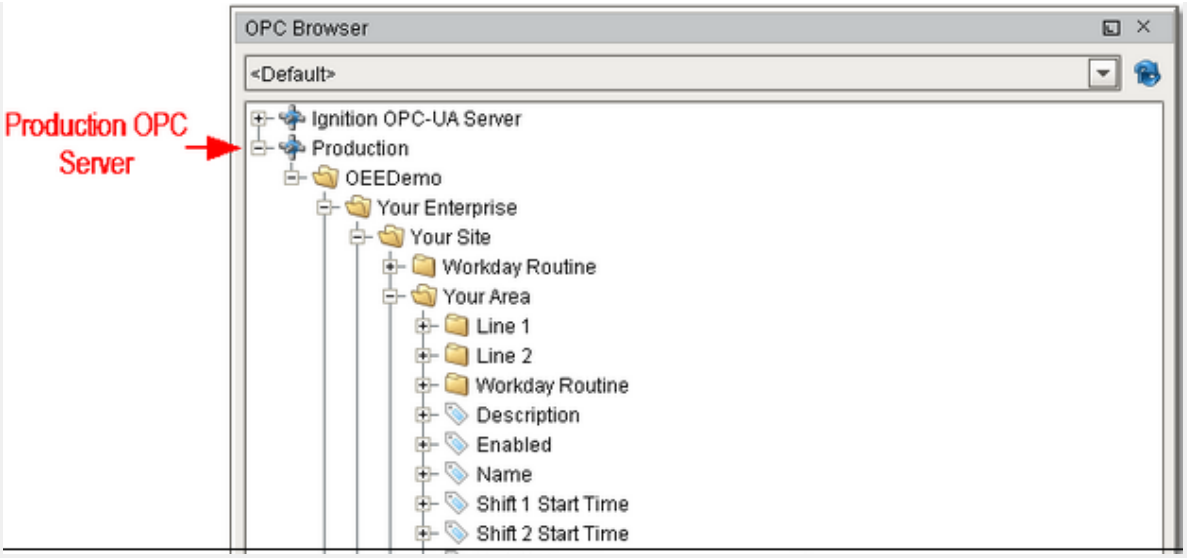
The general settings are accessed by selecting the desired production item and selecting the General tab.

Setting	Description
Enabled	By default, the production item is enabled. It can be disabled by un-checking the Enabled setting and saving the project. This will stop the track and trace, OEE, downtime, SPC, recipe and scheduling modules from using the area and any other production items that are underneath it.
Description	This is an optional description and is just for your reference.

OPC Production Tags

As production items are added to the production model, run time access into configuration settings and current state of those production items is available through the Production OPC Server. It is added automatically when MES Modules are installed. When the production items are added, removed or modified, the changes will be reflected in the Production OPC Server when the project is saved in the designer.

Please refer to the OPC Production Server Tag Reference in the Appendix for more help.

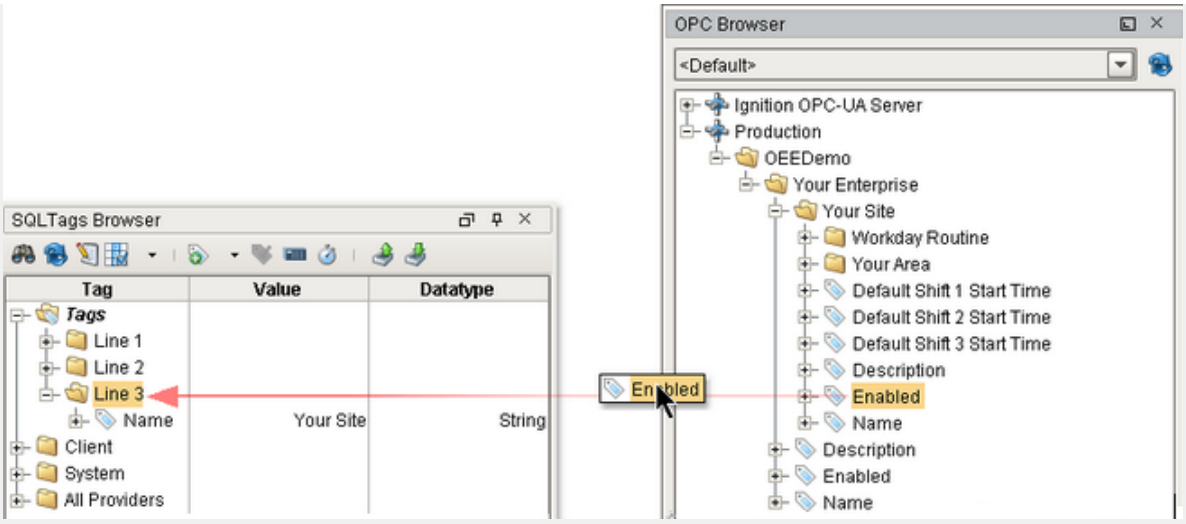


Demo OPC Values

Using OPC Production Tags in Your Project

Before Production OPC Server tags can be used in your project windows, transaction groups etc., they must be added to the Ignition SQLTags. This is done in the designer by selecting the SQLTag Browser and clicking on the OPC icon. This will cause the OPC Browser to appear. Next, drill down in the Production node within the OPC Browser. Drag the desired Production OPC Values over to the SQLTag Browser as shown.

When writing to OPC values that are related to production model settings, the new value is not retained upon restarting. This is because production model settings are saved in the Ignition project and is only saved when done so in the designer.



Add Production OPC Server Values to SQLTags

Using Supplemental Equipment

Mobile equipment that can be moved around such as bins, containers, die sets etc. cannot be defined in the production model as fixed equipment as defined by ISA-95. The MES Object Editor and scripting functions can be used instead to create Supplemental Equipment. Refer to Creating Process Segments - Supplemental Equipment for more details.

Configuring Lot Handling

Storage Units and equipment can be configured with different lot handling modes that define how lots (or batches) of material are stored at equipment. To change these settings, click on the Production Item in the Production Model and select the Trace tab.

Lot Handling Mode

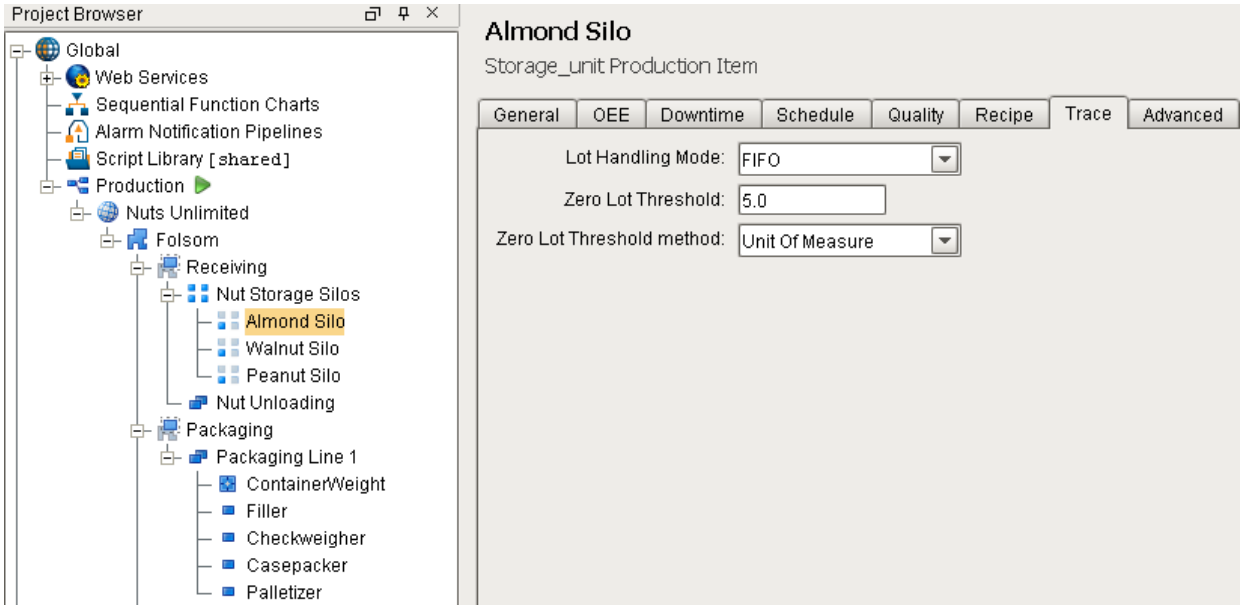
Valid types for Lot Handling Mode are ...

Mode	Description
Single Lot	Only one lot can be stored at this location even if the location is not full. e.g. Hopper that can physically hold only one lot
Random Lot	You can put any lots in any order and pull them out in any order. e.g. Storage Bin or Bay
FIFO	

Mode	Description
	This will cause the first lot stored at this location to be used first and the second lot to be used second and so on. e.g. Storage bin where you require the operator to use the oldest material first
LIFO	This will cause the last lot stored at this location to be used first and the second to last lot to be used second and so on. e.g. A location where you physically can't get to the material stored at the back
Same Lot	Material from the same lot that is received over multiple trucks or vessels can be added. e.g for lots larger than transportation method
Blend Lot	Allows only one lot at a time in that tank. If you add another lot, it will cause a new lot to be created. e.g. liquid Tank. See Tech note: How to Configure a Process Segment for Lot Blending

Zero Lot Threshold

The Zero Lot Threshold will cause the system to automatically zero the lot if not all of the lot was used from the silo. This cleans up a potentially large number of leftover lots with very small quantity remaining. The threshold value can be configured to be **Unit Of Measure** or **Percentage**.



Creating Material Classes and Definitions

MES Management Screen

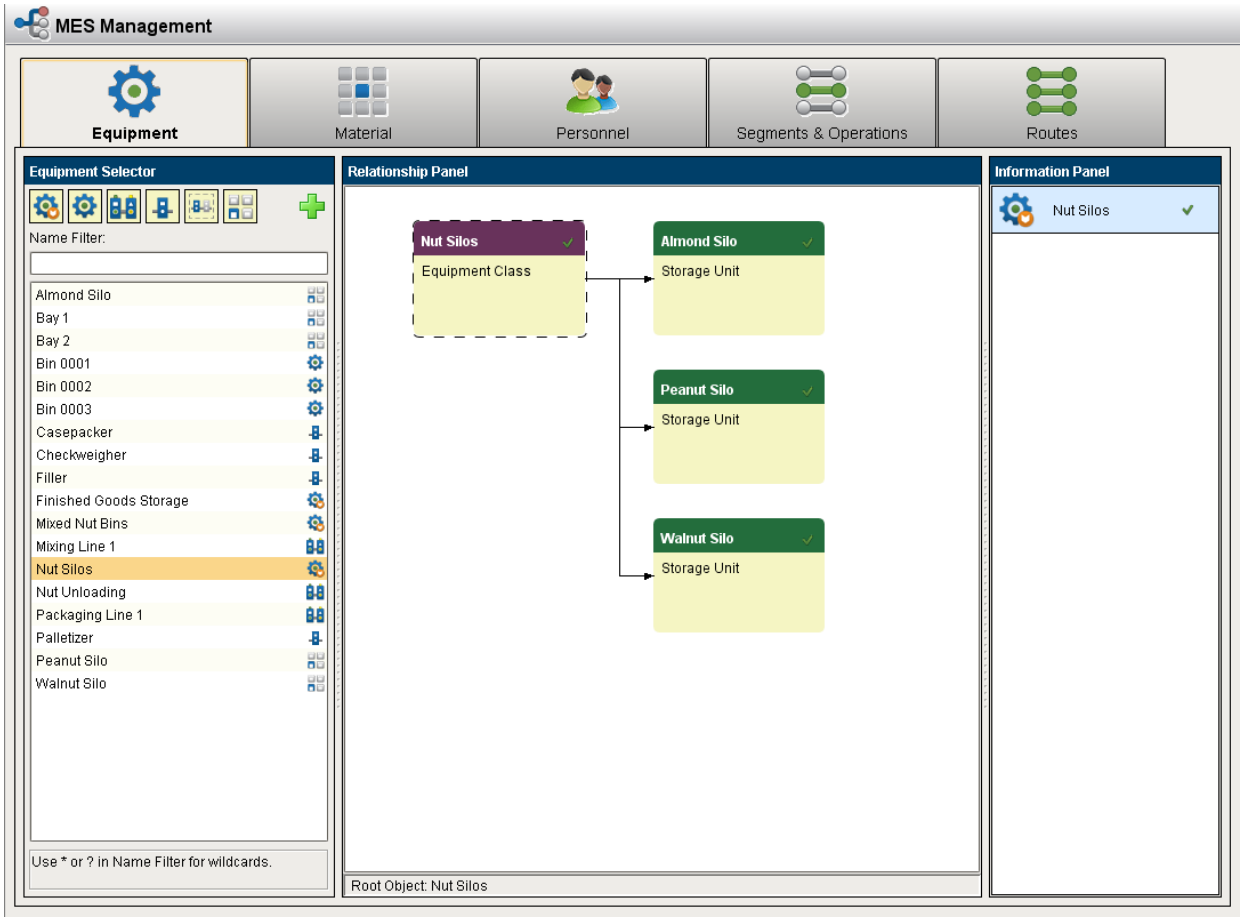
The MES Management screen provides a method for configuring Equipment, Material, Personnel, Segments & Operations, and Routes in the runtime client.

Based on the ISA-95 standard, Classes (or categories) can be created for Equipment, Material and Personnel.

The purpose for creating Classes of Material, or for that matter, Equipment or Personnel Classes, is that when an operation (process segment) is defined, we can define whether a class of material or a specific material can be consumed. This provides the ability to configure the Track & Trace implementation to add production control.

Scripting

The MES Management Screen can be used to create Material Classes and Definitions, however if a company has hundreds of materials or if the ERP system manages materials, it is likely that you will want to dynamically create materials through scripting. See KB article Creating Material and Supplemental Equipment through Scripting for more information.



Creating Process Segments

In the ISA-95 world, a process segment defines those plant capabilities used to execute production. In the Track & Trace module, we create process segments as a template, that can then be used by an operations definition/operation segment pair to create an operation that will be performed and data tracked against. The process segment defines where an operation can take place, who can perform an operation, what materials can be consumed and what materials will be generated. Process segments can be created and its properties edited through scripting or by using the MES Object Editor.

Process segments are never executed. They are a template for which operation segments are derived from.

Creating Segments with Scripting Functions

The following scripting function is provided that can be used to create process segments.

- system.mes.createMESObject

The following knowledge base articles show examples of how to use these scripting functions to create operation segments.

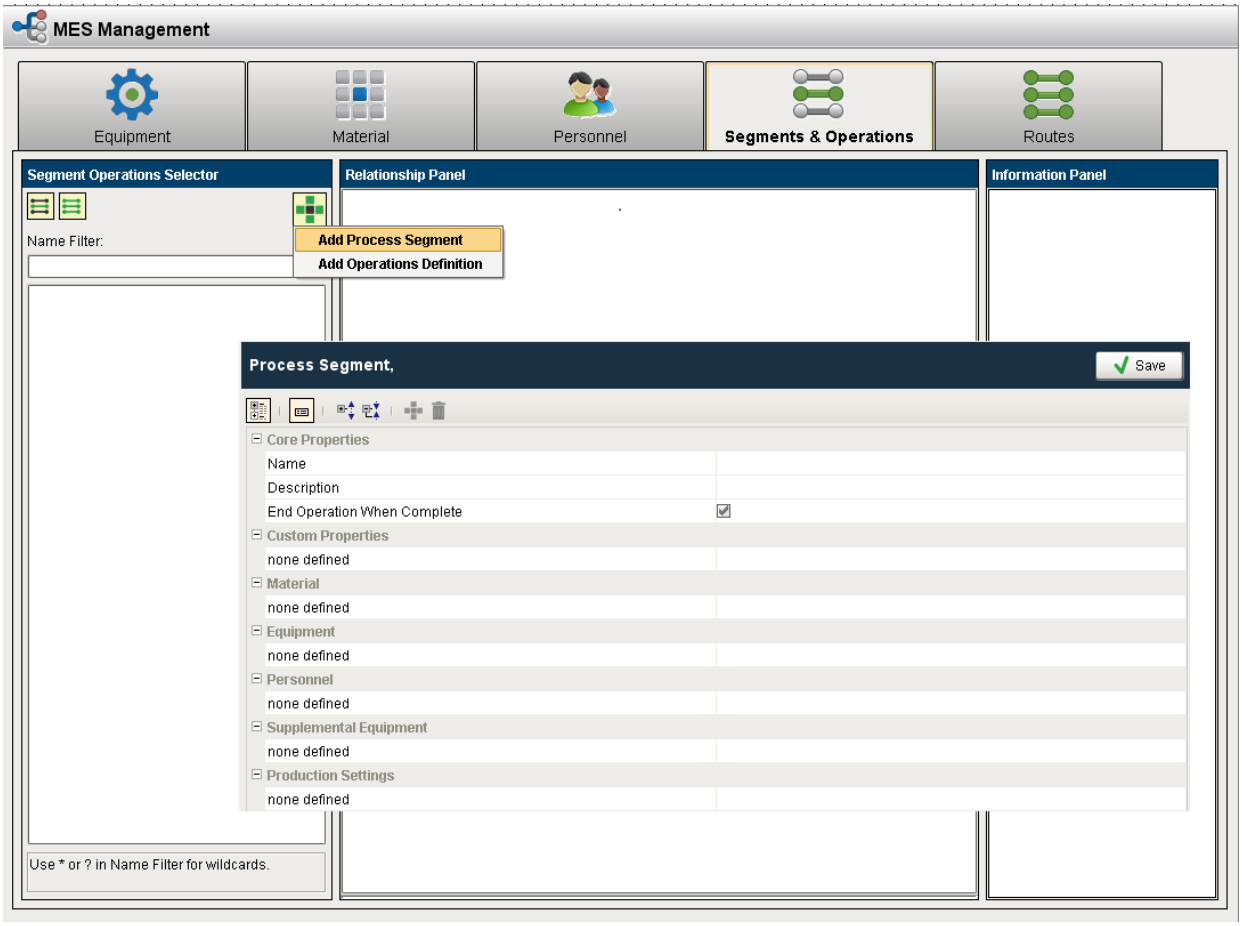
- Create Process Segment

The sections below detailing core properties, custom properties and complex properties are valid for creating process segments through both scripting functions and the MES Object Editor.

Creating Segments with the MES Object Editor

The MES Object Editor component provides a visual method for creating process segments and operations definitions. The **Editor Mode** property of the component must be set to **Segment Operations.**

Click on the  icon to select **Add Process Segment**. A popup will appear with sections grouped into **Core Properties**, **Custom Properties**, and what we call Complex Properties (**Material**, **Equipment**, **Personnel** and **Supplemental Equipment**).



Core Properties

Property	Data Type	Description
Name	String	Name of the process segment to be added
Description	String	Description of the process segment
End Operation When Complete	Boolean	If this property is set to True, then the operations automatically ends when segment is complete
Segment Recipe Name	String	This is a drop down list of available recipes which can be set to be used on segment start

Custom Properties

Custom properties can be added here for any meta data you want to store during execution of this process segment. Data stored here can be accessed through scripting for analysis. Custom properties are optional. Refer to setCustomPropertyValues for more details on using scripting functions to set custom properties.

Property	Data Type	Description
Name	String	The name of custom property
Value	String	The value of custom property
Units	String	The unit description of the custom property value
Data Type	String	The data type of the custom property. Valid options are **Int1**, **Int2**, **Int4**, **Int8**, **Float4**, **Float8**, **Boolean**, **String**, **DateTime**, **Int1Array**, **Int2Array**, **Int4arrray**, **Int8Array**, **Float4Array**, **Float8array**, **BooleanArray**, **StringArray**, **DateTimeArray**
Description	String	Description of custom property
Production Visible	Boolean	If this property is set to True, then the custom property will be visible at the time of production
Required	Boolean	This box is unchecked by default. If this setting is checked, then this property is a required property and if it is unchecked, this is an optional property
Custom Properties		To add a custom property of a custom property, select this row and click add button on tool bar

Complex Properties - Material

Any materials consumed or produced by this process segment are defined here. There is no requirement to create material complex properties e.g. a maintenance process segment, and there is no restriction on how many input, output or consumable materials are defined for a process segment. Refer to createComplexProperty for more details on creating complex properties with scripting functions.

Property	Data Type	Description
Name	String	The name of the material
Optional	Boolean	If the material defined here is always required as an input or an output of this process, then leave this setting unchecked. If this material is optional, then select it
Production Selectable	Boolean	This box is checked by default. If this setting is unchecked then this property selection will not be visible to the operator in the MES Material Selector component.
Use	String	Defines whether this material is an **input**, a **consumable**, **by-product** or an **output** of this process segment. Options are: • **In**: This setting is used for material feeding from an existing material lot into a segment that will be part of the finished goods. • **Out**: This setting is used for material feeding out of a segment that is or will be part of the finished goods. • **Consumable**: This setting is used for material feeding into a segment that is not part of the finished goods. • **By-product**: This is used for material feeding out of a segment that is not part of the finished goods.
Auto Generate Lot	Boolean	If this is set to True, the system will create a new material lot object for this material
Material Reference	Python Dictionary	This setting defines the type of material that can be used in this process segment. It can be a **Material Class** or a **Material Definition** and can be used to prevent the wrong materials being used
Lot Equipment Reference	Python Dictionary	This settings defines the location or equipment associated with the material lot. It can be an **Equipment Class**, **Equipment**, **Line**, **Line Cell Group** or **Storage Unit** and can be used to prevent this process segment from selecting lots located in other equipment.

Property	Data Type	Description
Enable Sublots	Boolean	If this setting is selected, then sublot support will be enabled for the material resource
Lot Number Source	String	This determines the source of the lot number. Options are: • **Auto**: This setting automatically generates lot number. The internal lot number generator will generate a lot number and assign it automatically for the operator. This option can also be used if a different lot number format is used or lot numbers are provided by another system that is integrated with this system. • **Manual**: This setting will prompt the operator for the lot number. This is typically used when receiving raw materials or entering a lot number generated by an outside system. • **In Link**: Only valid for **Out** materials. In cases where the lot number for the output material will be the same as the lot number of the input to the process segment, this setting will tie the two together. Segments can be configured with multiple material inputs and outputs and different lot number links can be configured. Whenever this setting is used, the **Lot Sequence** number is incremented, so that a material lot can be tracked through a number of process segments. **Creating Custom Lot Numbers** When the Lot Number Source is set to **Auto**, the T&T module will internally generate a unique lot number. If a custom lot number needs to be generated or a query to obtain a lot number from another system, a custom script can be added to the **CreateLotNumber** event to do just that. This event is fired when Lot Number Source is set to Auto and a new material lot number is required by an operation segment. See the Generate Custom Lot Number knowledge base article for more information.

Property	Data Type	Description
Lot Number Source In Link	String	The system will use the same lot number as the material reference with the name same as that of this setting. For example if this is set to **Material In**, then the system selects the lot number for the material reference **Material In**
Quantity Source	String	This setting determines the quantity of material. Options are: • **Available Lot Quantity**: Number of items belonging to the lot can be obtained with this feature. • **Link**: This option allows the quantity to come from an input or output material resource of this segment. This eliminates the need to type in the quantity multiple times if they will always be the same as another material resource. • **Link Combine**: For segments that are combining two or more lots into one output, this option can be used. It is used by having two or more material resources, that are segment inputs, linked to the same material resource output. When the segment is ended, the system will sum up the quantities of the linked material resources to that of the linking material resources. There isn't a need to use "," or any other delimiters while defining the quantity source link, instead a single name is used that can be put into all of the material references link names. • **Link Split**: For segments that are splitting a lot into two or more streams, as is the case of separating good from bad product, this option can be used. It is used by having two or more material resources, that are segment outputs, linked to the same material resource. When the segment is ended, the system will ensure that the sum of the quantities of the linking material resources equal that of the linked material resources. • **Manual**: The operator will be prompted for the quantity. The quantity of material can be entered manually or with script. It must be entered before the segment is ended.

Property	Data Type	Description
		• **MES Counter**: Obtain the quantity from the automatic production counters defined for the associated equipment. The associated equipment may change if the Lot Equipment Reference setting is set to a Material Class and the specific equipment is not known until the segment is started for production. More information can be found in the MES Counters page. • **Sublot Count**: The quantity will be automatically set based on the number of Material Sublot items belonging to the Material Lot.
Quantity Source Link	String	This is used when the Quantity Source setting is set to **Link**, **Split** or **Combine**. It is the name of the material resource to link to this segment. For an input Material, if Quantity Source is set to Available Lot Quantity or Manual, and the Quantity Source Link to **combine**, for the output material, set the Quantity Source to **Link Combine** and the Quantity Source Link also to **combine**
Quantity	Float8	Typically this is left blank, but it can be set to a fixed value that will be constant every time the segment is used for production
Units	String	Units for the quantity
Rate Period	String	This is used to set the material rate period. Options are: **Min**: For setting the rate in minutes. **Hour**: For setting the rate in hours. **Cycle**: For setting the rate in cycles.
Rate	Float8	This setting determines rate of the material. Material rate is invalid if it is set to less than zero
Cycle Time	Int4	The expected time to complete a material cycle in seconds
Final Lot Status	String	

Property	Data Type	Description
		When a segment is started, the status of the Material Lots will be set to Active. When the segment is ended or a new lot is used for the material resource, the status will be set to **Complete**. Optionally, the value of this setting can be used instead of the default **Complete**. This is useful for setting a lot to **Hold**, **In Process** or anything that can be used to filter lots or sublots. Active status while the lot is active cannot be changed.
Auto Lot Quantity Completion	String	This setting determines if all the items belonging to the lot are automatically consumed. Options are: • **Disabled**: Select this if the items in the lot should be used automatically. • **Enabled**: Select this if the items in the lot should not be used automatically. • **Confirm**: Set this if the l ot quantity needs to be confirmed. This will check if the lot has reached its confirmation threshold or not.
Material Lot Depletion Warning	Int4	Sets the depletion warning in seconds
Material Lot Status Filter	String	The material lot status can be set to any custom value here. This can be used to filter results when querying material lots
Custom Properties		To add new custom property, select this row and click add button on tool bar

Complex Properties - Equipment

A process segment requires one and only one Equipment complex property to be defined. this sets up production control on where this process segment can be executed. If you want to create a process segment that can occur on any piece of equipment e.g. 'Clean Equipment', set the equipment reference to an equipment class that contains all equipment. Refer to createComplexProperty for more details on creating complex properties with scripting functions.

Property	Data Type	Description
Name	String	The name of the custom property that can be used to reference this equipment resource in script
Equipment Reference	Python Dictionary	This is the Production Item (equipment) that the operation can be run at. This can be set to an equipment class or piece of equipment (Line, Line Cell, Line Cell Group or Storage Unit)
Use	String	The equipment use property defined here is in compliance with ISA-95 standard. We don't have any internal functions for this though you can specify if this property refers to Production, Maintenance, etc.
Quantity	Float8	The quantity of equipment defined here is in compliance with ISA-95 standard. We don't have any internal functions for this
Units	String	This specifies the units for the quantity setting
Custom Properties		To add a new custom property, select this row and click add button on tool bar

Complex Properties - Personnel

Personnel complex properties allow to you add production control to a process segment by defining who can execute this process segment. This complex property is optional and can be left blank. Refer to createComplexProperty for more details on creating complex properties with scripting functions.

Property	Data Type	Description
Name	String	The name of the custom property that can be used to reference this personnel in script

Property	Data Type	Description
Optional	Boolean	If the property defined here is always required, then leave this setting unchecked. If this property is optional, then select it
Production Selectable	Boolean	This box is checked by default. If this setting is unchecked then this property selection will not be visible to the operator in the MES Material Selector component.
Personnel Reference	Python Dictionary	This can be set to a Personnel Class or a Person. Setting this to Personnel Class will cause the operator to be prompted for the specific person for this personnel resource in the MES Material Selector component. If set to Person, then the selection will be automatically selected
Use	String	The personnel use property is here in compliance with ISA-95 standard, we don't have any internal functions for this though you can specify if this property refers to Production, Maintenance, etc.
Quantity	Float8	Personnel quantity property is here in compliance with ISA-95 standard, we don't have any internal functions for this
Units	String	This specifies the units for the quantity setting
Custom Properties		To add new custom property, select this row and click add button on tool bar

Complex Properties - Supplemental Equipment

Mobile equipment that can be moved around such as bins, containers, die sets etc. cannot be defined in the production model as fixed equipment as defined by ISA-95. The MES Object Editor and scripting functions can be used instead to create Supplemental Equipment. Refer to createComplexProperty for more details on creating complex properties with scripting functions.

Property	Data Type	Description
Name	String	The name of the custom property that can be used to reference this supplemental equipment in script

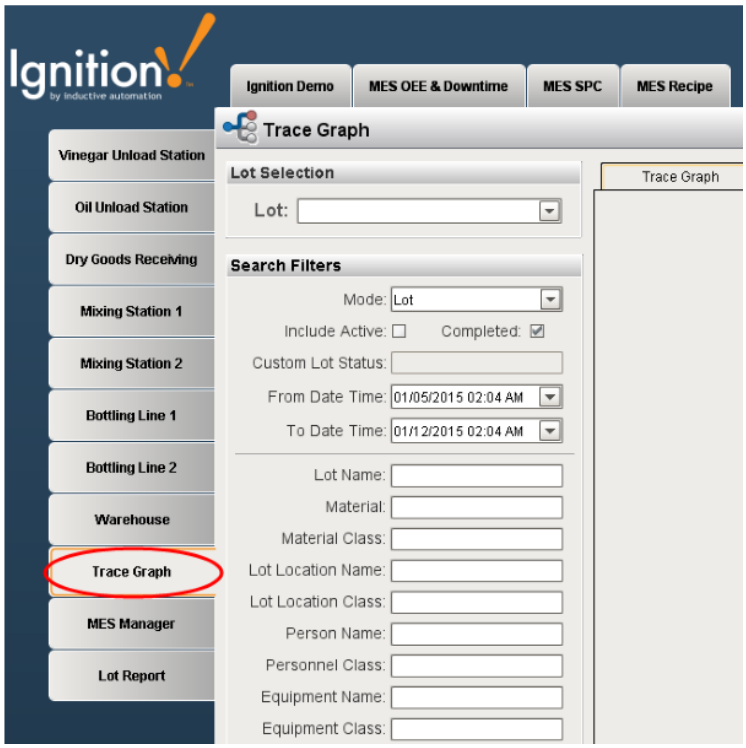
Property	Data Type	Description
Optional	Boolean	If the property defined here is always required, then leave this setting unchecked. If this property is optional, then select it
Production Selectable	Boolean	This box is checked by default. If this setting is unchecked then this property selection will not be visible to the operator in the MES Material Selector component.
Equipment Reference	String	The reference to this equipment
Use	String	The supplemental equipment use property is here in compliance with ISA-95 standard, we don't have any internal functions for this though you can specify if this property refers to Production, Maintenance, etc.
Quantity	Float8	The quantity of supplemental equipment is here in compliance with ISA-95 standard, we don't have any internal functions for this
Units	String	This specifies the units for the quantity setting
Custom Properties		To add new custom property, select this row and click add button on tool bar.

8.2.3 Traceability

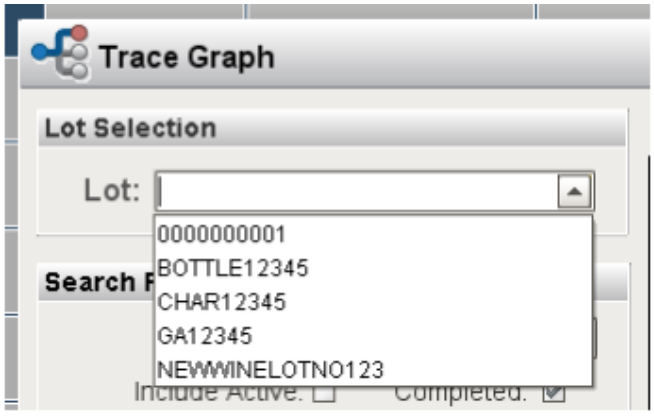
Trace Graph

The Trace Graph component provides a visual method to understand what material lots and processes were used to create Finished Goods. The Trace demo project includes a screen to view the trace graph.

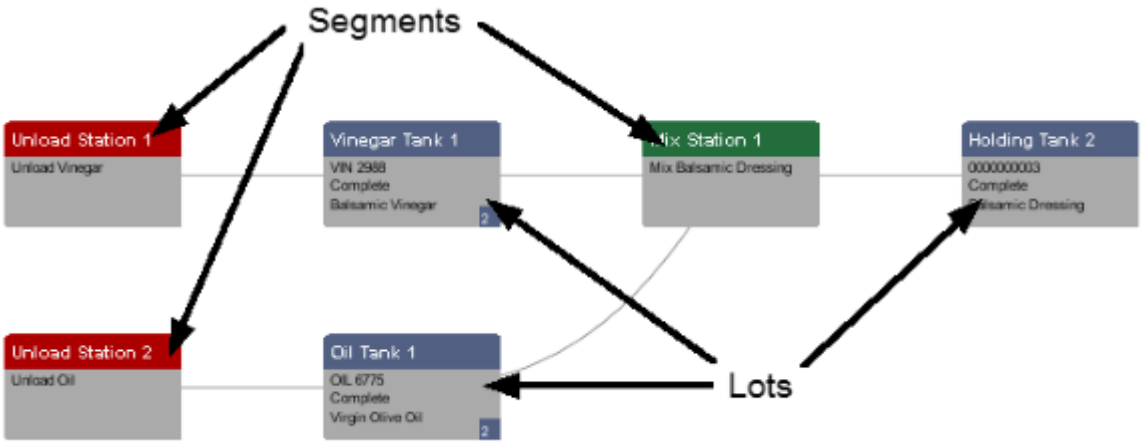
To use the Trace Graph simply select a From Date Time and a To Date Time to look up all of the lot numbers in that time period. You can filter for specific equipment or material and you can specify whether or not you want to see active or completed lots. Once you select your filters, you should be able to see a list of lot numbers in the dropdown list.



If you select Lot No 0000000001 and you will see the trace graph on the right.

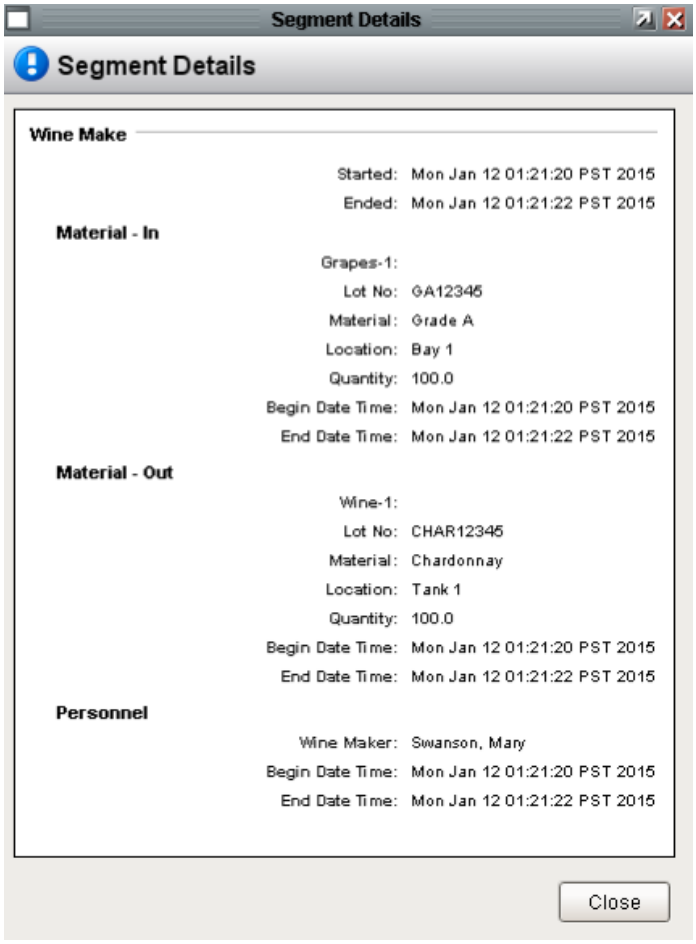


The nodes are laid out in chronological order from left to right. The node type alternates starting with a segment then showing a lot. The idea behind this is there are lots that are inputs to an operation and there are lots that the operation produced. In the image below, the upper left node titled Unload Station 1 is the operation that vinegar was unloaded. When this operation was done, a new lot VIN 2988 was created. Then that lot was used in the operation of making of balsamic dressing at Mix Station 1, which produced balsamic dressing that resides in Holding Tank 2.



Here is the trace graph for Lot No 0000000001.

You can click on any node to view the corresponding details. For example, click on the Wine Make and select Show Details. A popup window will open with the details.



Optionally, you can click on individual nodes to allow the trace to start inspecting that Lot No. This would give the details about the raw materials, finished goods, and much more. To view the data in tabular form, click on the Table tab. There are more details in the user manual about the trace graph component.

Components

The Track and Trace module provides a set of components that can be used to enter in data for an operation and to select material lots etc., for analysis. The demo project comes with quite a few screens pre-made that implement these components.

8.2.4 Material Lots and Inventory

Whenever an operation is performed, material lots are produced and consumed. The type of material lots that are consumed, the method in which they are consumed or produced, are defined in the material complex properties of the Process Segment. The Track & Trace Module provides scripting functions, binding functions and components that allow you to view material lots and inventory at any location.

Lot Sequence

Whenever an IN material lot is consumed by a process and the OUT Material is set to have the same Material Lot ID as the IN Material (by setting the **Lot Number Source** to **Link** in the material complex properties of the Process Segment), a unique incrementing Lot Sequence number is generated and associated to the new material lot. This provides the ability to search for a Material Lot and determine the most current Material Lot properties, such as location.

Objects

The Material Lot object can be used to return salient information regarding a Material Lot.

The MES Object Filter object can be used to return a list of Material Lots. See Filter for a Material Lot by Custom Property knowledge article for more information.

Scripting Functions

system.mes.getLotInfoByName

system.mes.getLotInfoByUUID

system.mes.getLotInventoryByEquipment

system.mes.getLotInventoryByLot

system.mes.getLotList

system.mes.getLotTraceByLotUUID

system.mes.getLotTraceBySublotName

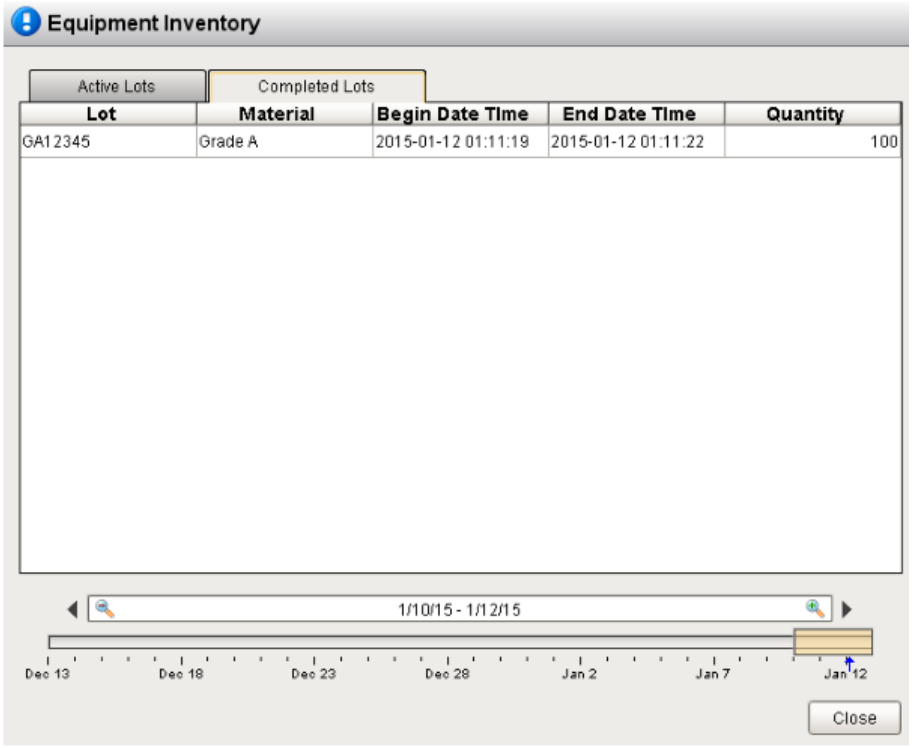
system.mes.getLotTraceBySublotUUID

system.mes.getSublotInfoByName

system.mes.getSublotInfoByUUID

system.mes.loadMaterialLot

system.mes.lot.filter.createFilter



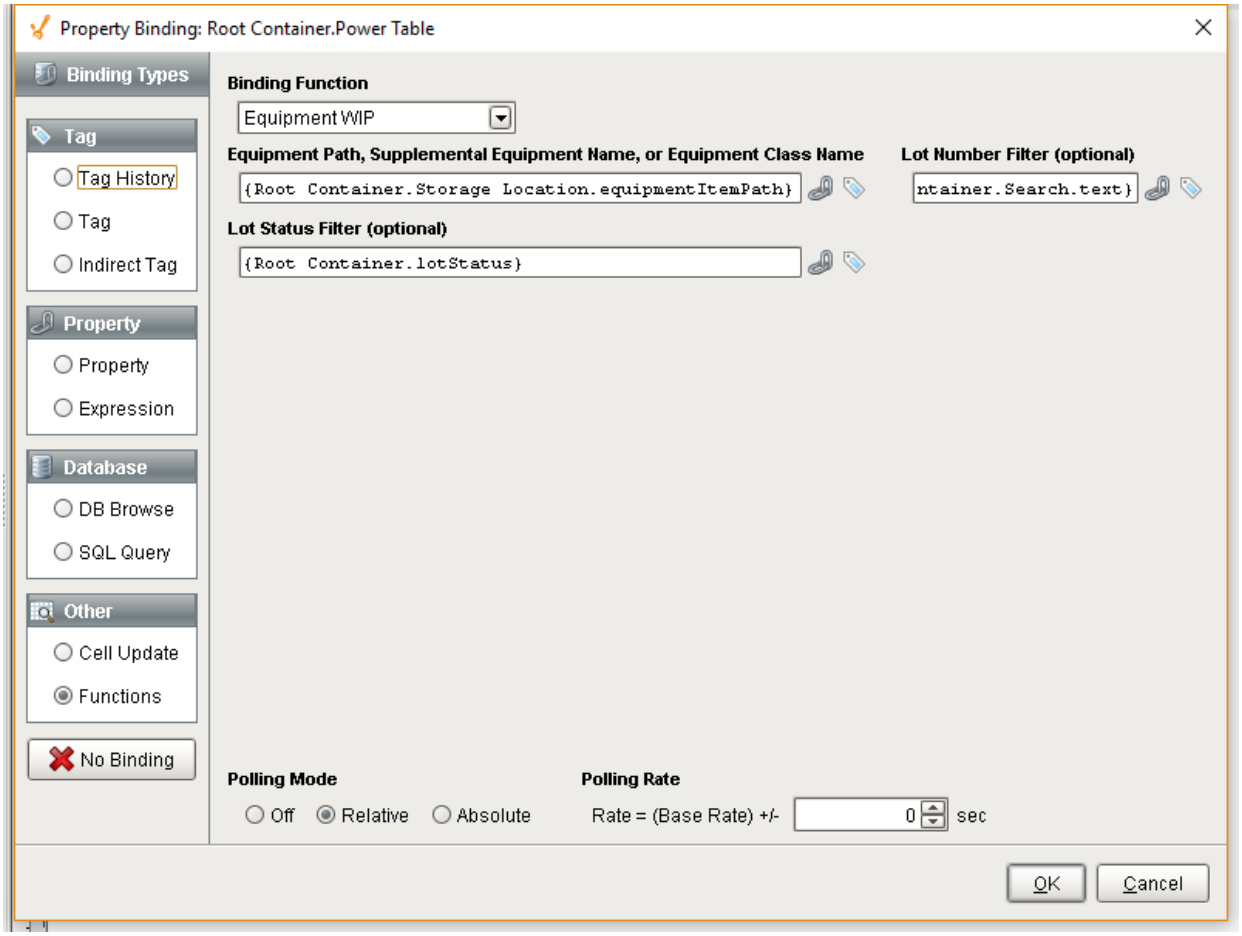
Binding Functions

For more information refer to the Trace Binding functions section in the reference appendix.

Components

MES Material Selector

MES Sublot List



8.3 SPC

New to SPC?

Download and testdrive this most powerful MES solution available anywhere!

Download and Install Module

A Simple Workflow

Step 1. Database Connection

Step 2. Configuring MES Databases

Step 3. Installing the Production Simulator

Step 4. Production Model Configuration

SPC Module in a nutshell

Click on the topic you would like to learn more about ...

- Components
- Scripting
- Objects

Product Data Sheet

To see the Product Data Sheet,

click on SPC Module

Click here to see Knowledge base articles of **SPC**.

Info

Sepasoft SPC module uses the following functions for scripting:

system.quality.spc

system.quality.definition

system.quality.sample.data

Read this section about licensing...

Licensing and Activation

8.3.1 What Is SPC

SPC is the acronym for **Statistical Process Control**. When we are testing our process, we need to have an understanding about the attributes that defines the quality of the specific product. Then we have to go through a process of understanding of why the test failed. SPC is more specific around the actual test that we do.

- Collecting samples
- Analysis of the test data.

- Rule violations - conditions where we are out of control, out of our range.
- Quality goes on planning how often taking a test

SPC is a tool used to manage quality.

What Is Quality

In manufacturing, quality is defined as a measure of excellence or a state of being free from defects, deficiencies and significant variations. It is brought about by strict and consistent commitment to certain standards that achieve uniformity of a product in order to satisfy specific customer or user requirements. Quality products help to maintain customer satisfaction and loyalty and reduce the risk and cost of replacing faulty goods.

Your customers expect you to deliver quality products. If you do not, they will quickly look for alternatives. Quality is critical to satisfying your customers and retaining their loyalty so they continue to buy from you in the future. Quality products make an important contribution to long-term revenue and profitability. The quality of a product is very important for a stable manufacturing industry. The defected or the faulted goods must be sorted out from the good ones. So it is extremely important to maintain the quality of any manufacturing product. This is why we have introduced the SPC module. This module ensures the quality of items and increase the productivity of efficient products.

Quality Versus SPC

Although Quality and SPC are sometimes used interchangeably, they are different. Quality is very broad and includes much more than just SPC, while SPC can be considered a tool in the Quality process.

A quick example may help to point out the difference. If product in the warehouse is going bad over time, then a process has to start to narrow in on the cause. It will involve brainstorming, perhaps creating fishbone diagrams to determine the possible causes. In the case of an off-color product, it could be rust building up in pipes, chemical formulation changes, or different raw materials being used. This part of the example refers to quality. Unlike SPC, quality requires more than installing software, collecting samples and analyzing the results.

Once the most likely causes of the off-color product have been determined, SPC can be used to monitor the attributes and narrow down and isolate the cause. It may be determined that when the pH of a sub-ingredient falls out of a certain range, the stability of the product color is degraded. With this knowledge, SPC can be used to monitor the pH so that if it falls outside of range, it can be corrected quickly. This prevents a bigger problem that may arise when the product stays in the warehouse for a period of time.

SPC Variation

All manufacturing processes are affected by intrinsic variation. Variation exists in everything. No matter how hard we try, there can never be two identical actions that generate exactly the same result. Too much variation leads to rework, scrap, or customer problems. As the variation in our processes is reduced, the output of our processes will be improved. That’s our goal with SPC - to reduce the variation in our processes and then monitor the process to make sure the variation doesn’t increase. So first of all, we have to make a frequency tally of the data. Next is to calculate it's normal distribution of measurement values.

SPC Causes of Variation

When a manufacturing process involves complex machines to complete production, a temporary malfunction or a breakdown in an intricate piece of equipment can affect the manufacturing process. Identifying means of improving efficiency of all working parts of production promotes a continual and more efficient operation. Positioning of equipment and the personnel required to operate machines can also affect production.

Environment
The climatic conditions to which the commodity was exposed before receipt; what conditions are likely to occur whilst commodities are held in storage.

Raw materials
The availability of materials and the development of new, hi-technology materials will have an influence on the final design of a product. Quality of the finished product always depend on the quality of raw materials.

Methods
Quality also depends on the methods used to produce it and the chemicals added during production. To maintain high standards of quality, companies are investing in new machines and following new procedures and methods these days.

SPC Samples

Sometimes it is tedious to measure every part and so we should have to go for sampling. Sampling is the process of selecting units from a population of interest so that by studying the sample we may fairly generalize our results back to the population from which they were chosen. Instead of checking every product, We just take measurements on some of them in random.
Every hour we may pull out some samples to represent the population. We shall define the sample size, which is basically equal to the number of measurements. Sepasoft SPC defines this with the following.

- Interval

- How many measurements, which is your sample size.

SPC Values and Attributes

There are some attributes of items that we can't measure. Such as missing logo, cracks etc. After a specific shift of may be 30 days, we may have 100 cracks, 5 missing logo. In other words, multiple defects per item.

SPC Standard Deviation

In statistics, the standard deviation (SD, also represented by the Greek letter sigma, σ for the population standard deviation or s for the sample standard deviation) is a measure that is used to quantify the amount of variation or dispersion of a set of data values. In other words it is the amount that we are deviating.

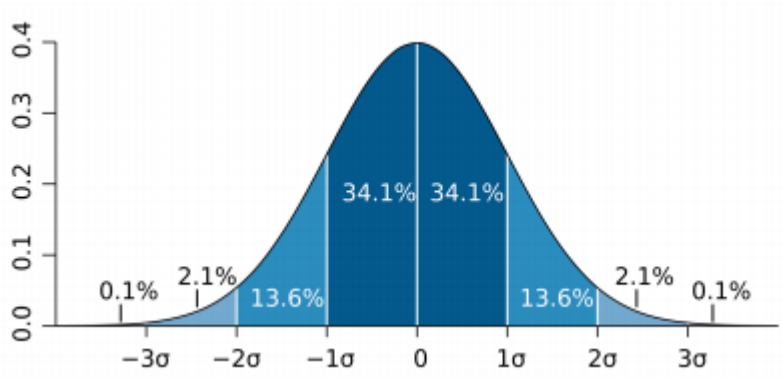
Standard deviation of sample means is calculated with the following equation:

$$\sigma_{\bar{x}} = \frac{\sigma_X}{\sqrt{n}}$$

where x is the standard deviation of individual measurements and n is the sample size. The Upper (UCL) and Lower (LCL) control limits are calculated by the following equations:

$$UCL_{\bar{x}} = \bar{X} + z\,\sigma_{\bar{x}} \text{ and } LCL_{\bar{x}} = \bar{X} - z\,\sigma_{\bar{x}}$$

where the z value is the number of standard deviations (sigmas) from the mean to put the control limits.



8.3.2 SPC Module Overview

The SPC Module provides a number of features that can be used to implement the policies and procedures of a Quality Management System (QMS). It is not in itself a QMS. What it does is exceed the capabilities of normal SPC software applications by providing...

- Manual Sample Collections

- Automatic Sample Collections
- Scheduling of Samples based on real-time production conditions
- Alerting of Samples Coming Due, Due or Overdue
- Automatic Evaluation of Control Limits and Out of Control Signals without human intervention
- Alerting of Out of Control Conditions
- Customizable Screens

...and much more

The SPC module in itself provides some very powerful capabilities, and with it sitting on top of the Ignition platform, allows for significant configuration and customization to realize your desired functionality.

This module can be used to ensure that statistical data is accurately collected on time, every time, helping you eliminate any issues leading to quality problems. It is an industry-standard methodology for measuring and controlling quality during the manufacturing process. Quality data in the form of Product or Process measurements are obtained in real-time during manufacturing. This data is then plotted on a graph with pre-determined control limits. **Control limits** are determined by the capability of the process, whereas **specification limits** are determined by the client's needs.

Automatic Sample Collection

SPC data can come from a variety of sources so the SPC module provides you with the tools you need to collect it. Sepasoft SPC was built with support for automatically collecting data from PLC devices, OPC-connected devices, lab instruments, RS232 devices, USB devices, data files, web services and external databases. When automatic data collection is not possible, the SPC Module supports manual data entry. In cases when there is no network connectivity, offline data collection is possible using mobile iOS or Android devices.

Lab Instruments

For legacy laboratory inspection equipment that does not provide an OPC-UA or other interface, the Instrument Interface module allows for data collection through the parsing of data files or by intercepting data on an RS-232 COM port. See the Instrument Interface help section for more information.

Additional Factors

Sepasoft SPC supports the collection of data not typically associated with quality, but that can directly impact it. These factors can include raw material vendors, maintenance, humidity, temperature and much more.

Scheduling Samples

If you worry about samples being taken at the correct time and not being faked after the fact, you are not alone. It is not a matter of whether or not the person responsible for taking samples has been distracted and missed taking samples, it is a matter of when. The Sepasoft SPC module has powerful features that will schedule samples based on current real time production conditions.

For example, if a lab staff is required to take samples every hour a production line is running, what happens when there is a break down or the production start is delayed because the lack of raw materials? How does the lab technician know when production started and if it has been an hour? In a variety of ways, the Ignition module can let the lab technician know that production has started and a sample is coming due, is due or is overdue. This can be expanded to instantly inform all parties that should know of various sample due states.

This can be utilized for more than taking live process samples. It can also be used for other checks that have to be done around the production facility such as weekly inspections of values or rodent traps.

Simple Automatic Sample Scheduling

Taking accurate, regularly-scheduled samples is vital to maintain quality. That's why we made it easy to schedule samples automatically in real-time using the Sepasoft SPC Module.

You can take samples based on actual production conditions and use convenient, automated alerts to help ensure that samples are never missed. Customizing a sampling schedule is simple and totally flexible, empowering you to accommodate the requirements of your unique production environment.

Notifications

The Sepasoft SPC Module makes it easy to ensure that samples are taken on time. A samples list shows upcoming, due and overdue samples by department or location so you can quickly see what's happening. The system can also automatically send out email and SMS notification or flash screen indicators to prompt sample taking. You even have the option of setting up the system to automatically stop production if a sample is overdue.

Sample Approval

When samples are taken, the SPC Module can be set up to automatically approve them or hold them for approval. This feature can help you ensure that samples are approved when and how you need them to be.

Sample Definitions

You can easily create sample definitions to define attributes, control limits, signals and sampling locations. Choose attributes from variable data types such as real, integer, Boolean and many more.

Automatic Alerts for Out-of-Control Conditions

Typically, SPC software requires that someone opens a screen and visually checks for out of control conditions. Just like the scheduling of samples, someone may be distracted by other pressing production issues and fail to complete the task. The Sepasoft SPC module has powerful features that will automatically evaluate out of control signals every time new sample data is recorded. This can be expanded to instantly inform all parties that should know of various out of control conditions.

Quick Automatic Signal Evaluation

To ensure quality, it's important to keep processes within acceptable control limits. This is done by identifying out-of-control variations as soon as possible without human intervention. Every time a sample is taken, the Sepasoft SPC Module evaluates out-of-control conditions and automatically alerts you if they are present. With Sepasoft SPC, find out-of-control variations quickly and resolve them before they become a bigger problem.

Out-of-Control Alerting

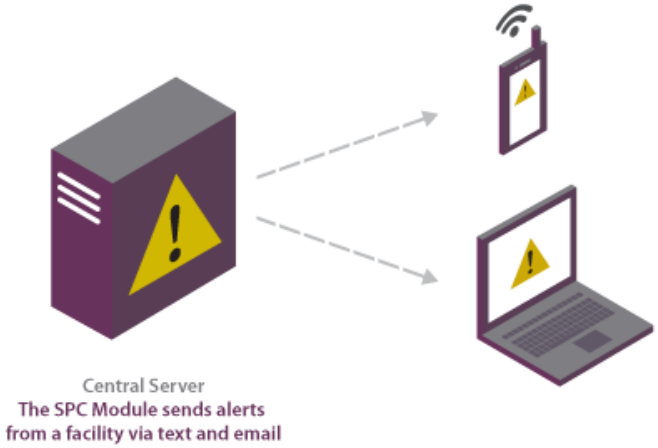
The system alerts you automatically whenever an out-of-control condition or bogus sample data has been detected. Alerts are easy to customize so you can have Sepasoft SPC send out an email or SMS notification, flash a screen indicator, stop production, or do a number of other alert methods.

Control Limits

The Sepasoft SPC Module has typical built-in control limit calculations. These typical control limits can be modified or added to simply by editing, copying or creating new ones.

Signals

To get you up to speed faster, the SPC Module is provided with standard signals. But when the production environment calls for something more, you can edit, copy or create your own signals.



Analyze SPC Data with Customizable Charts

All the data in the world is of no use without good analysis tools, so we built the Sepasoft SPC Module with a full range of powerful and flexible SPC Control Charts. Based on security roles, control limit values can be calculated and set interactively on the Control Charts. The Additional Factors feature gives you the flexibility to associate and visualize other production information along with SPC data and with customizable appearance settings for charts, tables, control limits and signals, you have the power to see the information you need in the way you want.

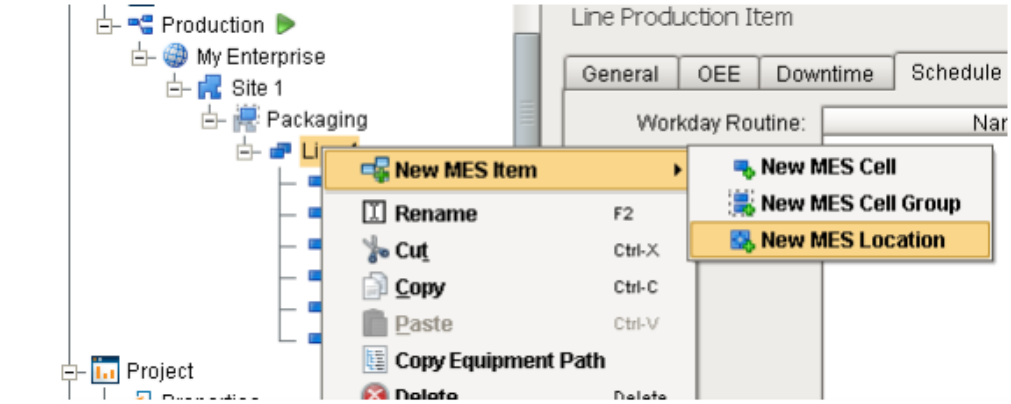
The control charts can be separated into three groups: value charts, attribute charts, and analysis charts. On all charts, it is possible to add assignable causes and notes to explain a data point. A sample note can be entered on the Lab or Test Stations page when the sample is first entered. This can be done by selecting a sample, then clicking Add Note. An attribute note is added directly from an SPC chart by right-clicking on a data point and selecting Set Note from the drop-down list. In addition to attribute notes, an assignable cause can also be added in this way. Assignable causes can also be saved for future use. Out-of-Control Signals and Control Limits can also be added to the graphs.

8.3.3 Production Model Configuration

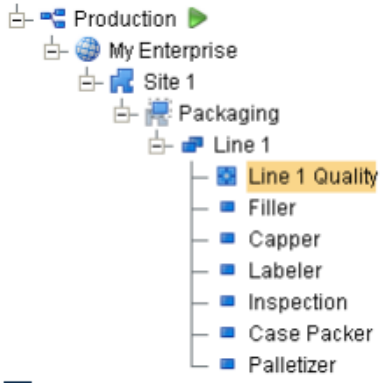
The SPC module deals with MES Locations. Locations are where samples are taken. Samples can be taken automatically or manually. Regardless, a location must be added to the production model before taking a sample. Locations can be added to an area or a line.

Locations are added to the Production model in the Ignition Designer. Locations can be added at the Area and Line level.

You can add as many locations that you take samples at. Keep in mind you may have multiple samples you take at a particular location. For example, you may only take 1 sample on the line whereas you take 3 samples in the lab. In that case you would just need 2 locations in the production model.

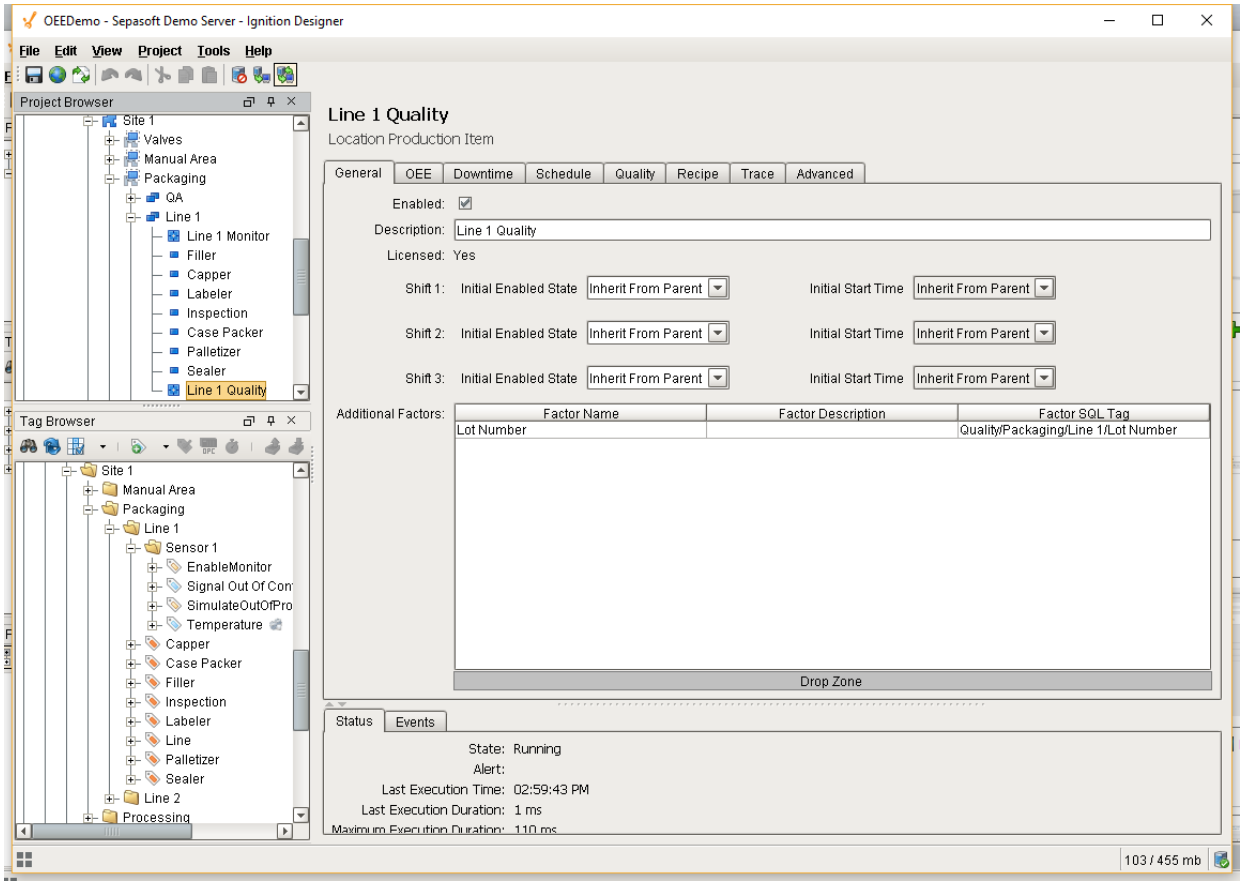


Rename the location to "Line 1 Quality." The icon is different that a MES Cell.



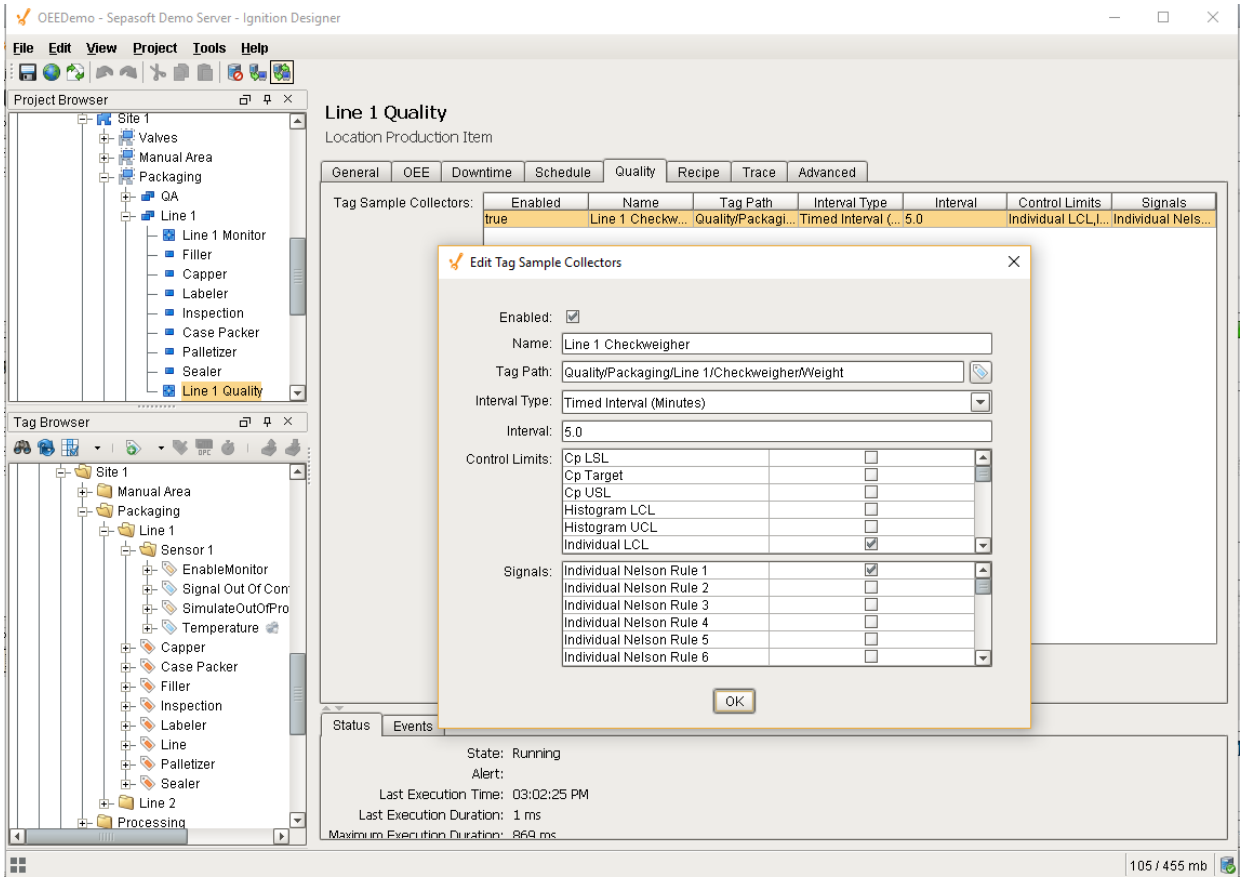
Additional Factors

You can setup Additional Factors on the General tab of the location production item in the Production model. Any defined additional factors will have there value stored and associated the sample whenever it is taken. This provides a flexible method of extending the SPC engine to associate meta data with the sample.



Sample Tag Collectors

The Quality tab of the location production item provides the ability to define automated tag sampling. For more info, refer to the Automated Tag Sample Collector.



Setting Up Your Tag Structure

It is a good idea whenever using any of the MES modules to layout your tag structure to mimic that of the Production Model.

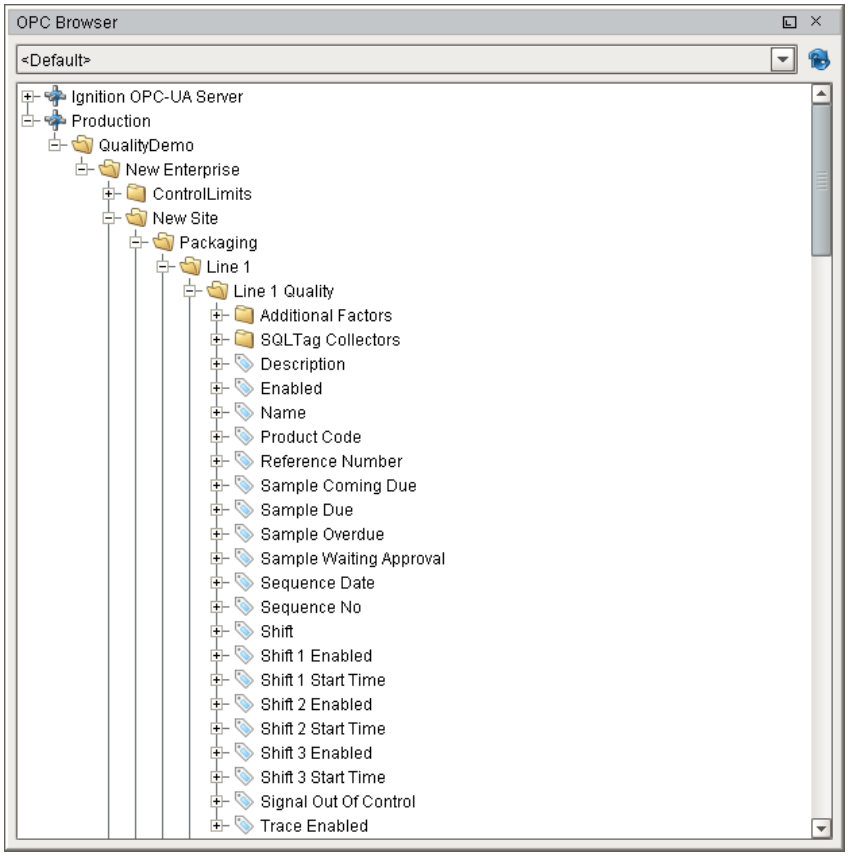
OPC Production Tags

The SPC module will create the following OPC Production server tags for each Location that is defined in the Production Model.

- Location
- Intervals
- Signals
- Control Limits

The production model is defined in the Ignition designer and contains your production areas, lines and locations. Access to the configuration and current state of the production model is available through the Production OPC Server. It is added automatically when the SPC Module is installed. When the production items are added, removed or modified, the changes will be reflected in the Production OPC Server when the project is saved in the designer.

The image on the right shows some of the values available to read, and in some cases write to. For more information, see Appendix A



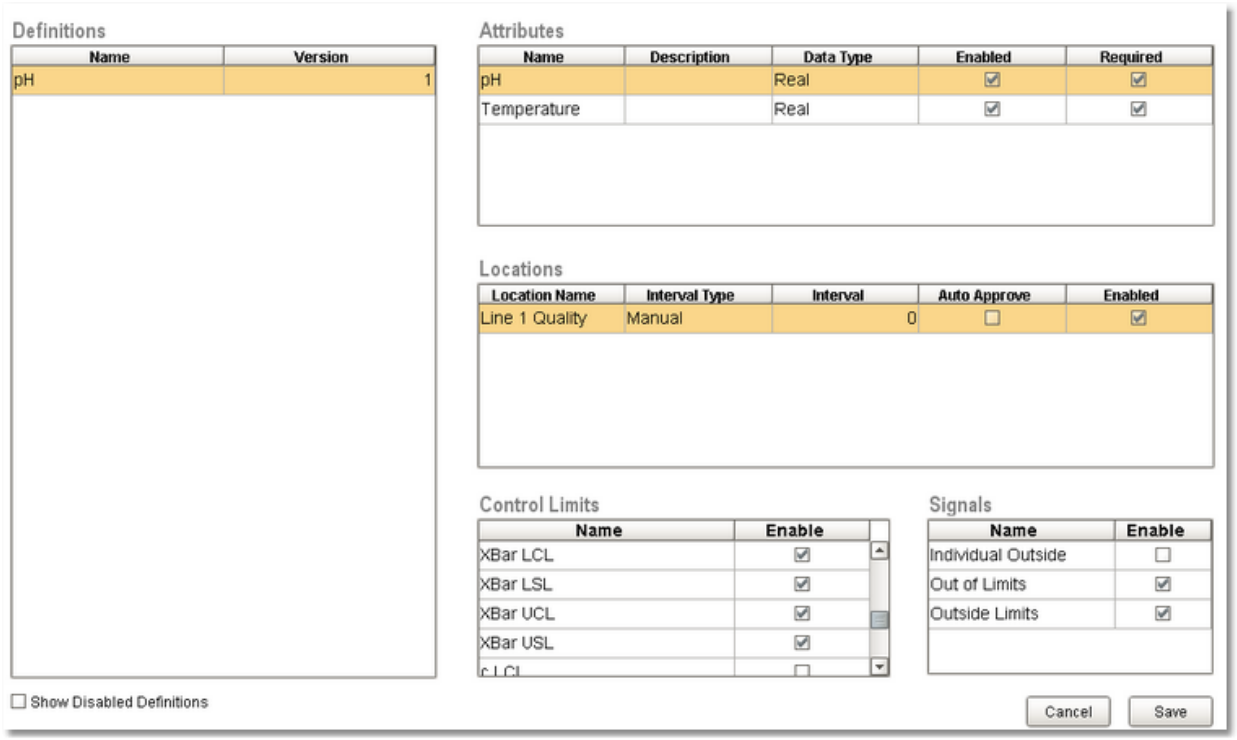
SPCOPCBrowser

8.3.4 Sample Definitions

Sample definitions originate from two different sources. One source is the Tag Sample Collectors that are defined in the designer and are for the sole purpose of creating samples automatically from Ignition tags (no human intervention). The other source is from the sample definitions created using the screens covered in this section, and are for the purpose of manual or semi-automatic collection of sample data (human intervention).

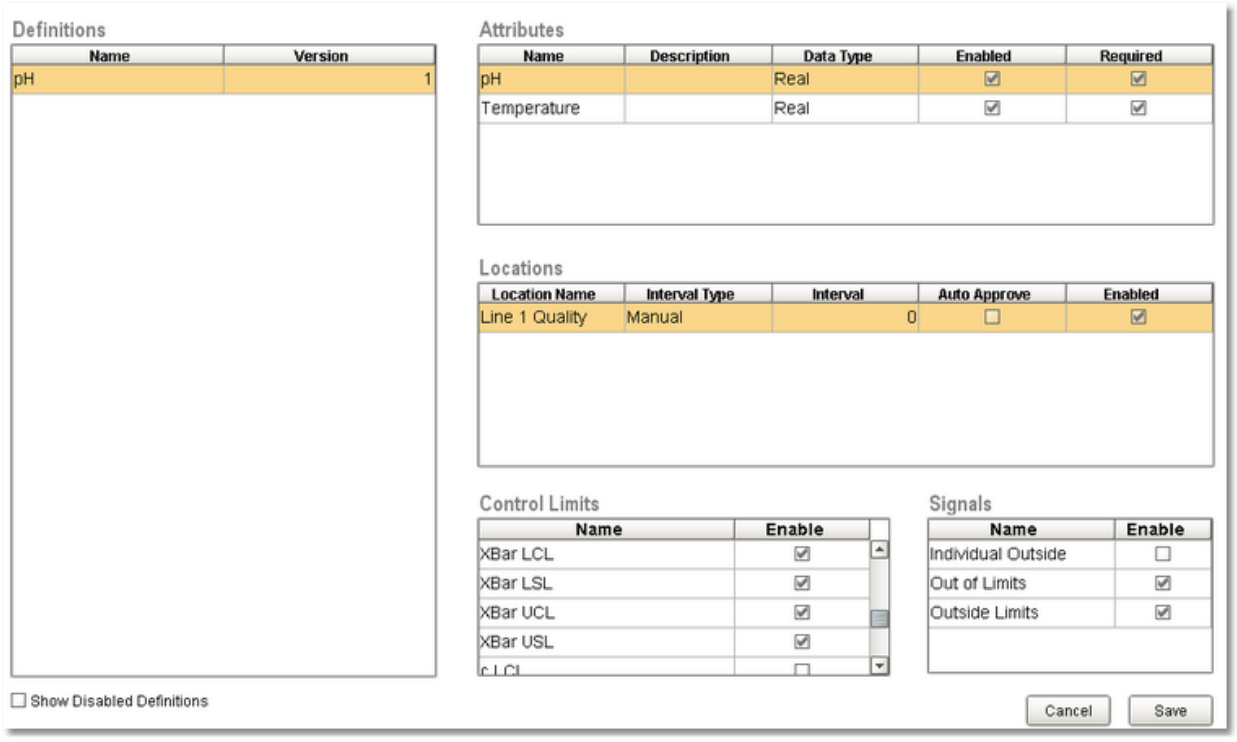
Sample Definition Manager

The sample definition screen is made up of components from the SPC modules that work together to allow for the management of sample definitions. By selecting a sample definition, the attributes, locations, control limits and signals associated with it are shown. The attributes define the data measurements to collect for each sample. The locations define the virtual locations that are appropriate for this sample definition. The Control Limits table defines which limits to apply to this sample definition. And last, the signals define which out of control signals to apply to the sample definition.



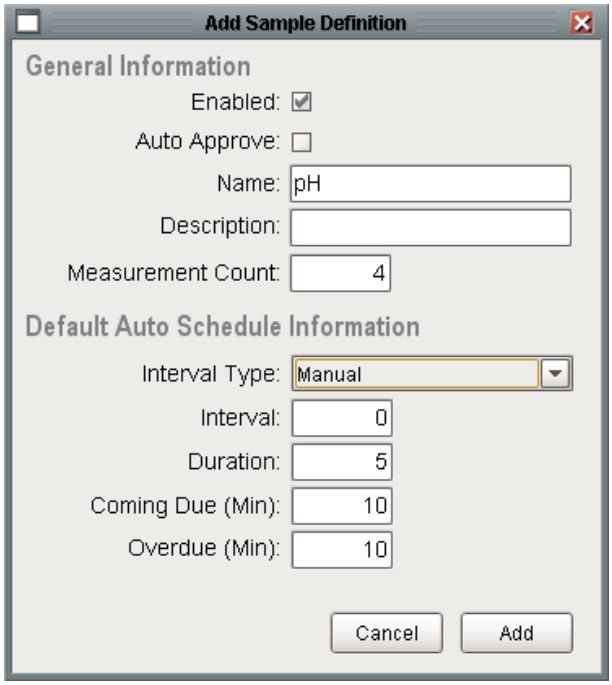
Sample Definition Page

Sample Definitions Screen



Sample Definition Page

Sample Definitions Screen

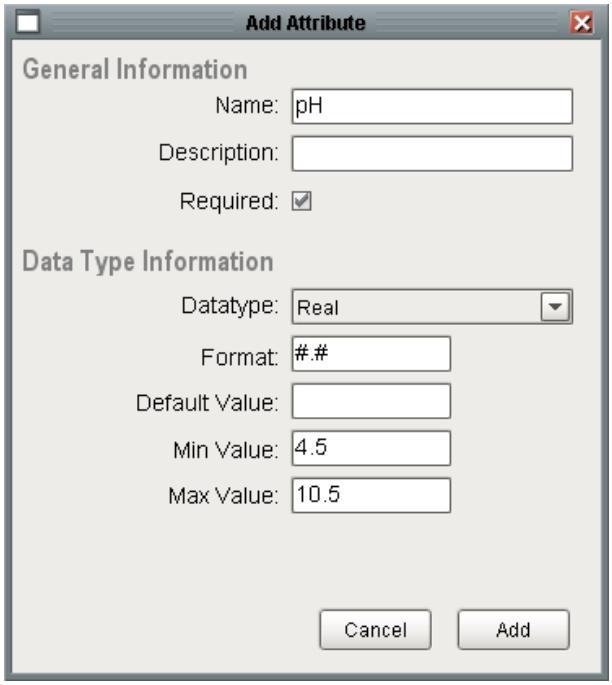


AddSampleDef

Add Sample Definition Window

Adding Attributes

After adding a new definition, the attributes must be defined. This is done by right-clicking the Attributes table and selecting **Add** from the drop-down menu. This opens a window similar to the one before, which allows users to define each attribute. Some examples of attributes include pH, temperature, viscosity, weight, nonconformities, and nonconforming items. From here, the name, description, datatype, format, default value, minimum value, and maximum value can be defined. This window also allows the users to decide if the attribute will be required when entering sample data on the Lab or Test Stations screen.



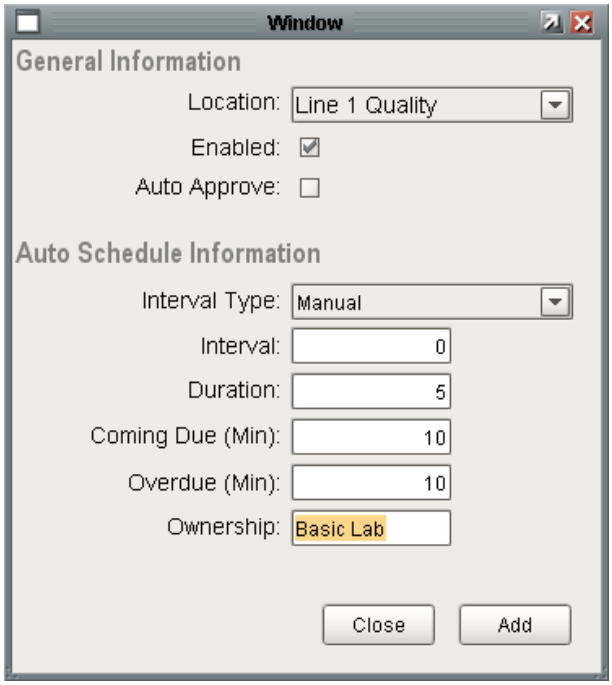
Add Attribute Window

Add and Edit Attribute Windows

Adding Locations

Next, the locations, or where the samples will be taken, can be defined. Again, this can be done by right-clicking on the Locations table and selecting **Add** from the drop-down menu. The ownership field declares who is responsible for the testing of the sample, whether that be the lab or the operator at the testing station.

The interval type defines how the samples will automatically be scheduled. Or as in the image below, they will be manually created by the user. If the interval is set to Timed Interval (Hours) then a sample will automatically scheduled as defined by the Interval setting. When a new project is created, the default Intervals options are also created but they can be modified, added to or even removed. See Sample Definition Location for more details of each of the settings.



Add Location Window

Add Location Window

Ownership

Sample definitions can be configured to specify an owner group who is in charge of that particular sample. The ownership is separate from the location and is used in filtering.

Adding Control Limits

Any selected control limits will be available to include on the control charts and will also be included in the automatic evaluation of out of control conditions of the sample data. When a new project is created, the default control limit options are also created but they can be modified, added to or even removed. Keep in mind that each control limit is associated with a particular control chart. For example, XBar UCL is associated and can only be used with the XBar chart. This is because the calculation used to determine the XBar UCL value is specific to only the XBar chart.

Control Limits

Name	Enable
Range UCL	☐
StdDev LCL	☐
StdDev UCL	☐
StdDev XBar LCL	☐
StdDev XBar UCL	☐
XBar LCL	☑
XBar LSL	☑
XBar UCL	☑
XBar USL	☑
c LCL	☐
c UCL	☐
np LCL	☐
np UCL	☐
p LCL	☐
p UCL	☐
u LCL	☐
u UCL	☐

Control Limits Table

Adding Signals

Any selected signals will be available to include on the control charts and will also be included in the automatic evaluation of out of control conditions of the sample data. When a new project is created, the default signal options are also created but they can be modified, added to or even removed. Keep in mind that each signal is associated with a particular control chart. For example, Individual Outside is associated with, and can only be used with, the Individual chart. This is because the calculation and control limits used to determine if a sequence of individual values are out of control is specific to the Individual chart.

Signals

Name	Enable
Individual Outside	☐
Out of Limits	☑
Outside Limits	☑

Signals Table

After all the desired settings have been defined, the user can select **Save** to commit all the changes, or **Cancel** to undo any changes that have been made. After a sample definition has been created, samples based on them may appear or be manually added depending on the Interval setting.

Adding Samples through Scripting

The SPC module provides components to add samples, however, there are situations where you might want to add the sample yourself. The SPC module also provides scripting functions to perform a variety of tasks such as adding a sample. Open up the Ignition designer and create a new main screen in the root of the quality project called **Add Sample Scripting**.

Drag a Button component from the Buttons tab of the component palette onto the window. Set the text of the button to **Enter Sample**.
Double click on the button to configure the **actionPerformed** event. Enter in the following script:

```
location = "[global]\My Enterprise\Site 1\Packaging\Line 1\Line 1
Quality"
sampleDef = "Measurement"
sample = system.quality.sample.data.getCreateSampleByName('',
sampleDef, location)
sample.setSampleData(1, "LocationX", "15")
sample.setSampleData(1, "LocationY", "24")
sample.setSampleData(1, "Diameter", "0.26")
sample.setApproved(1)
system.quality.sample.data.updateSample(location, sample, 1)
```

The first argument of the setSampleData function takes the measurement count which is 1 if there is only 1 measurement. Make sure you add a row for each attribute on the setSampleData function.

Of course we hard-coded the values in the script. They can come from the window or tags. You can make your own forms to add samples.

8.3.5 Automatic Tag Sample Collectors

Tag Sample Collectors are used to automatically collect measurement data from an Ignition tag and create samples with the collected measurement data. The sample will be a single measurement of a single attribute (value). When configuring, the selected interval defines how often to create a new sample. For example, on every 100th value change of a checkweigher value, create a new sample and record the current value. Or, every 10 minutes while a process is running, create a sample and record the current temperature. The measurement data can come from a variety of sources including any OPC connected device, values from external databases, manual entries, etc.

Any samples that are automatically created and recorded by a Tag Sample Collector are automatically approved and will appear in the control charts. By setting the Auto Refresh property of either the SPC Selector or SPC Controller components on client screens, new samples will appear in the control charts in real time as they are created. In addition, the appropriate events found on the Advanced tab for the production location will be executed.

Adding and Editing Tag Sample Collectors

To add a Tag Sample Collector, right-click the Tag Sample Collector table and select New from the drop-down menu. A window will appear with several fields to be completed, including the name of the tag sample collector, as well as the tag path and other properties required.

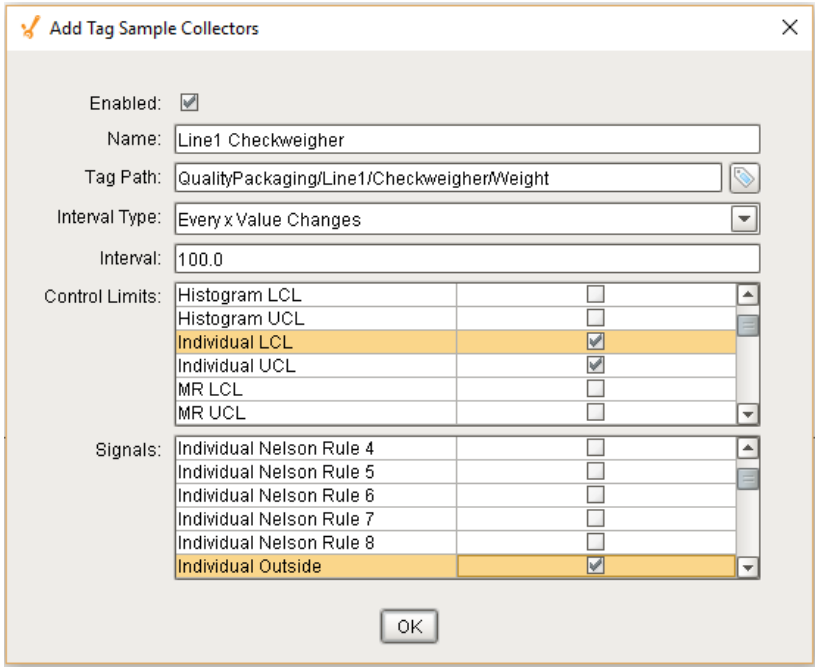
To edit a tag sample collectors, right-click the Tag Sample Collector table and select Edit from the drop-down menu. A window as shown below will appear identical to the window used to add tag sample collector. Once the desired fields have been edited, select OK.

Enabled

Tag Sample Collectors enabled property provides a method of stopping the automatic collection of measurements and creation of samples. Additionally, any tags associated with this property can be changed to start and stop automatic collection. See OPC Tags for more information.

Name

This is the required unique name of the Tag Sample Collector as it will appear, with **SQLTag-** prepended to it, in selection lists. Behind the scenes, a sample definition is created using this sample name. Sample definitions created for the purpose of Tag Sample Collectors will not appear in the definition management and manual sample entry client screens.



Add SQL Tag Sample Collector

SQLTag Path

This is the SQLTag path from which measurement values will be read.

Interval Type

The interval options that can be selected here match those defined in the Intervals list on the Enterprise quality tab. Only intervals that have script will be included as options for Tag Sample Collectors. The reason for this is that manual intervals, which are the those without script, will never be created and do not apply to automatic collection of measurements.

Interval

The interval to collect data and create new samples. The units of this interval are defined by the interval type and can be minutes, days, every x value read, etc.

Control Limits

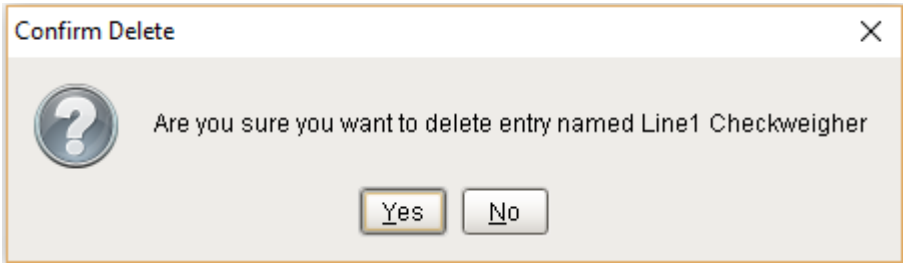
The control limits that are checked will be calculated for this Tag Sample Collector during signal evaluations. Available control limit options are defined in the Control Limits list on the Enterprise quality tab. It is important to include control limits that a signal depends on or the signal will not be evaluated correctly.

Signals

The signals that are checked will be evaluated every time a new sample is recorded by the Tag Sample Collector. Available signal options are defined in the Signals list on the Enterprise quality tab.

Deleting Tag Sample Collectors

To delete a tag sample collector, select the item to be deleted. After selecting, right-click the item and select Delete from the drop-down menu. A window as shown will appear confirming that you permanently want to delete the tag sample collector.



Exporting and Importing Tag Sample Collectors

To export tag sample collector entries, right-click anywhere on the table containing tag sample collector entries and select the Export menu item. A dialog box will appear to allow selection of an existing file or the entry of a name for the new file to which the collector entries are saved. If a file extension is not entered, then the default .csv will be used.

```
Enabled,Name,SQLTag Path,Interval Type,Interval,Control Limits,Signals
"true","Line 2 Checkweigher","Quality/Packaging/Line 2/Checkweigher/Weight","Every x Valu
```

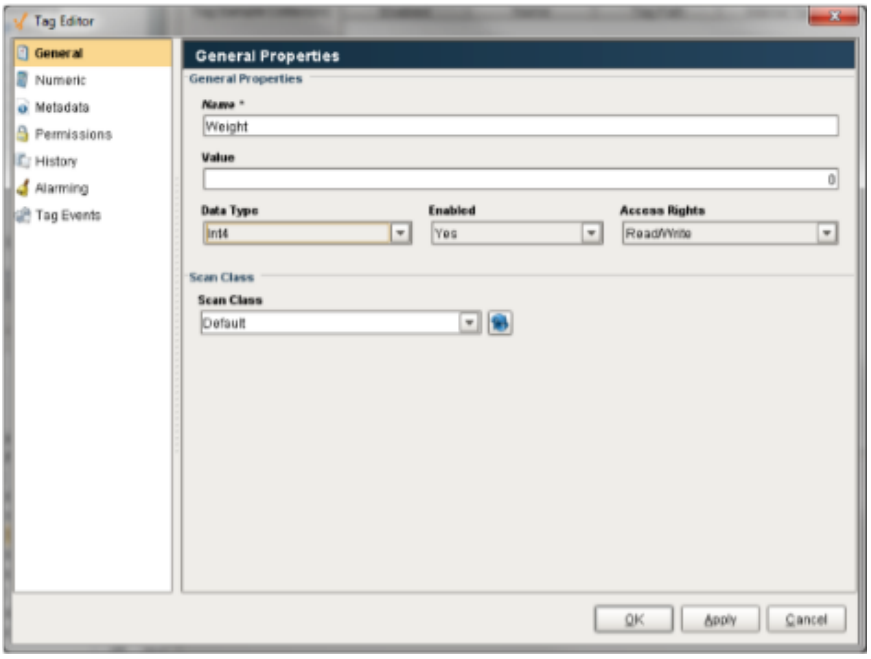
The first line of the file must at least contain the property names separated by commas. If additional names exist, they will be ignored. The property names can be in any order. Below is a sample csv file showing multiple tag sample collector entries. The lines in the example shown below have been shortened.

To import tag sample collectors, right-click anywhere on the entries and select the Import menu item. A dialog box will appear as shown below to allow selection of a comma separated values (csv) formatted file.

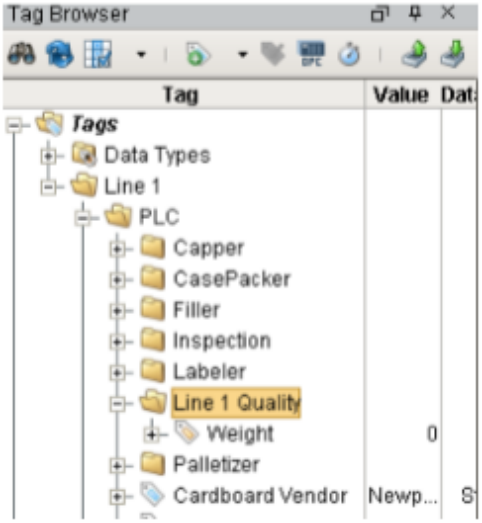
Example - Creating an Automatic Tag Sample Collector

Tag Sample Collectors can be added, edited or deleted on the Location page of the designer under the **Quality** tab. Let's add a checkweigher automatic tag sample collector to our **Line 1 Quality** location. In the Ignition designer select the **Line 1 Quality** location and select the **Quality** tab on the right. First, we have to create a tag that will be our value for the sample.

Let's use a memory tag so we can control the value. In our tag database add a folder called **Line 1 Quality** in the **Line 1 > PLC** folder. Create a memory tag called **Weight** that is an **Int4** with an initial value of 0.



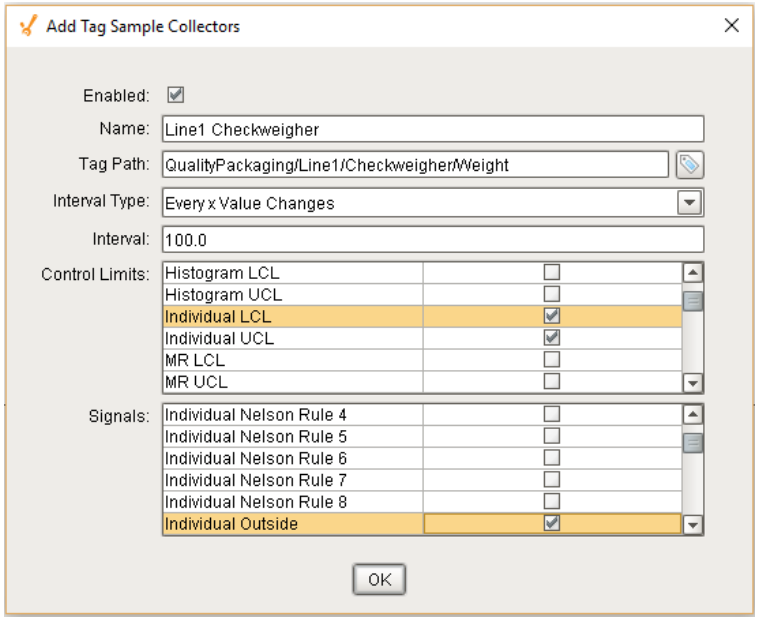
The tag provider should look like this.



Now that we have a tag let's create the collector. Right click on the **Tag Sample Collectors** table to add a new automatic tag sample collector. Let's call the collector **Checkweigher**. Locate the **Weight** tag we just created for the **Tag Path**. There are 3 important settings: Interval Type, Control Limits, and Signals. The Interval Type is used to determine when the sample is taken. It could every 10 minutes or every 100 value changes. The SPC module comes with several built-in intervals and it is possible to create your own. Set the **Interval Type** to **Every x Value Changes** and set the **Interval** to 5. That way Ignition will take a sample every 5 value changes of the tag. You can enable any control limits you plan on using for this sample

and you can also enable any out of control signals you plan on using. Select the **Individual LCL** and **Individual UCL** control limits. Select the **Individual Nelson Rule 3** and **Individual Outside** signals. That way we can be notified if the value falls outside of the LCL and UCL limits and if there are 6 consecutive points in increasing or decreasing order.

Press OK to save. Make sure to save your changes in the designer as well. As soon as you press save in the designer the samples will start to automatically get taken based on the interval type. For us, we have to change the value of the Weight tag 5 times for a sample to get taken. That's it! You can add as many automatic tag sample collectors as you need.



8.3.6 Control Limits

Statistical tables have been developed for various types of distributions that quantify the area under the curve for a given number of standard deviations from the mean. These can be used as probability tables to calculate the odds that a given value (measurement) is part of the same group of data used to construct the histogram.

The prominent statistician Walter Shewhart found that control limits placed at three standard deviations from the mean in either direction provided an economical trade off between the risk of reacting to a false signal and the risk of not reacting to a true signal, regardless of the shape of the underlying process distribution.

What Are Control Limits

Control limits , also known as natural process limits , are horizontal lines drawn on a statistical process control chart, usually at a distance of ±3 standard deviations of the plotted statistic from the statistic's mean.

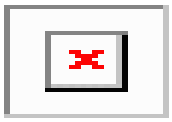
They should not be confused with *tolerance limits* or *specifications,* which are completely independent of the distribution of the plotted sample statistic. Control limits describe what a process is capable of producing (sometimes referred to as the 'voice of the process'), while tolerances and specifications describe how the product should perform to meet the customer's expectations

Control limits are Upper Control Limit (UCL) and Lower Control Limit (LCL) values that are calculated from the data that is gathered from a process. They are shown as horizontal lines on the control charts and reflect the past performance of that process. They can be either calculated or entered manually, or through scripting, to act as specification limits. Specification limits are requirements made by the company, not a reflection of the process itself.

For the p and u Charts, the control limits can vary for each sample depending on the number of items inspected for each sample. See the SPC Charts for more information.

Calculating Control Limits

At the start of an SPC implementation all efforts must be aimed at controlling the variation of the dispersion (via a range or sigma control chart). When the dispersion of the process is out of control, the average of the process is likely also out of control. This is because there is a relation between the variation in the dispersion and the variation of the average as shown in this equation:



equation1

Problems on the dispersion chart should be addressed first. When control limits on the average chart are calculated in the standard way (three sigma from the average), out-of-control variables on the dispersion chart will also lead to out-of-control factors on the average chart. These out-of-control factors on the average chart are not an indication of changes in the process average, however, but are a logical result of changes in the dispersion. When operators without adequate knowledge of SPC see out-of-control variables on both average and dispersion charts they will likely start working on the problem on the average chart because such problems are easier to address – just adjust the process.

The way to avoid this issue is to begin charting without putting the limits of the average chart at three sigma. There are three options:

1. Do not use control limits for the average – only show the target. This is sometimes called a run chart.
2. Fix the limits at a level which will rarely lead to out-of-control variables.

3. When the process average is unstable use modified control limits to minimize the actions, but make sure that process averages that are abnormal are signaled.

When the C_p value is high enough, the third method is preferred because the limits are still calculated based on the process variation. This method still gives an early warning when a disturbance of the process average will lead to defective products.

Control limit Types

There are different control limit types for each type of control chart. For example, the **XBar** control limit type only supports **XBar UCL**, **XBar LCL** and **XBar Other** control limits, and cannot be calculated or shown for any other control chart besides the XBar Control Chart.

Default Control Limits

The control limits are defined by the enterprise and can be added, edited or deleted on the Enterprise page in the designer under the **Quality** tab.

When a new Enterprise Production Item is added, the following control limits are added:

Kind	Description	LSL	Target	USL	LCL	UCL		Chart
Cp	Process Capability. A simple and straightforward indicator of process capability	*	*	*				Process Capability
Histogram	Histograms are used to show distribution of variables				*	*		Histogram
Individual	Individual Control Charts f or time series tracking of a process				*	*		Individual
MR					*	*		Individual and Range

Kind	Description	LSL	Target	USL	LCL	UCL		Chart
	Moving Range used to indicate process variation by calculating the ranges of two or more consecutive samples							
Median	Median is the middle point when data points are arranged from high to low				*	*		Median and Range
Pp	Process Performance . A simple and straightforward indicator of process performance	*	*	*				Process Performance
Range	XBar Range used to indicate process variation by calculating the ranges of two or more samples that have multiple measurements				*	*		XBar and R
StdDev	Standard Deviation				*	*		XBar and S
XBar	Xbar represents the sample mean of a number of repeated observations	*		*	*	*		XBar and S
c	Count type data, Total number of non-conformities (defects)				*	*		C-Chart

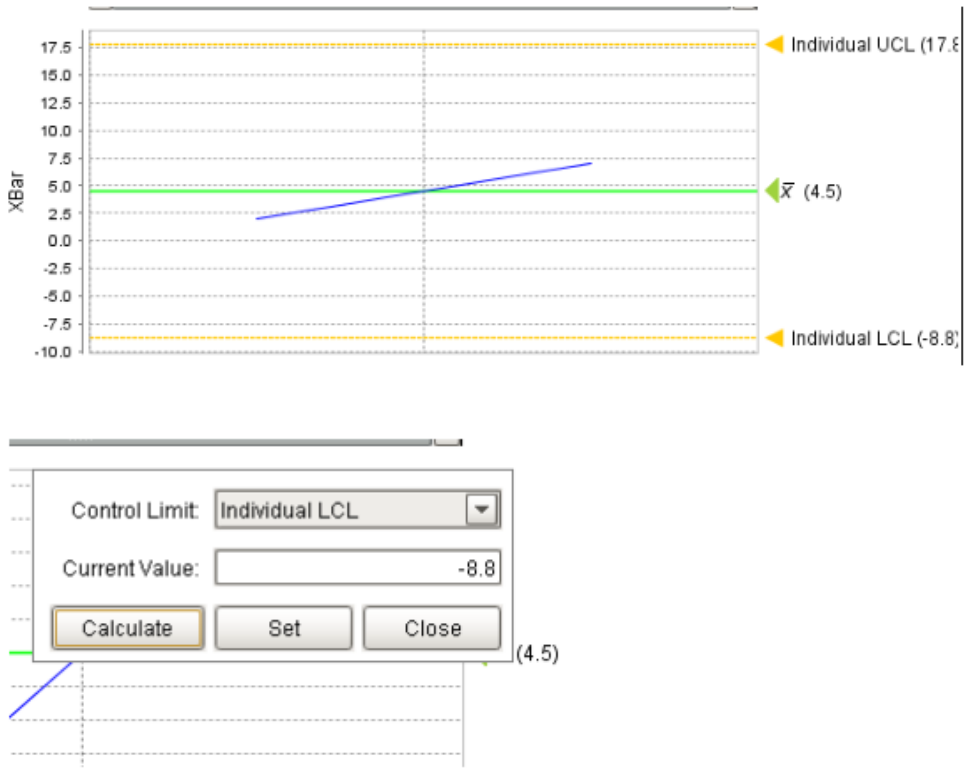
Kind	Description	LSL	Target	USL	LCL	UCL		Chart
	per unit or total number of events occurring in a given unit of time							
u	Count type data. Total number of non-conformities (defects) per item or group of items				*	*		U-Chart
np	Non Performing. N umber of nonconforming units in a sample				*	*		NP-Chart
p	Proportion Non Performing. proportion of nonconforming units in a sample				*	*		P-Chart

Adding and Editing Control Limits

To add a new control limit, right-click the Control Limits table on the Quality tab at the Enterprise level of the Production Model and select **New** from the drop-down menu. A window will appear with several fields to be completed, including the name and kind of the control limit, as well as the scripting necessary to use the control limit.

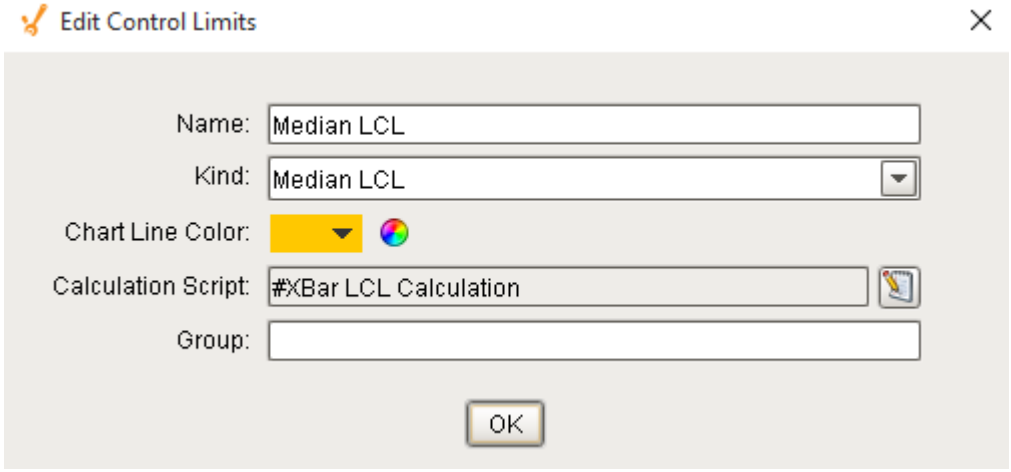
To edit an existing control limit, right-click the Control Limits table on the Quality tab at the Enterprise level of the Production Model and select **Edit** from the drop-down menu. A window will appear as shown. Once the desired fields have been edited, select OK.

This section refers to how to create Control Limits to be used to monitor sample values. Actual Control Limit values can be set through the control charts by right clicking on the LCL and UCL or through scripting. Refer to the knowledge base article on Setting Up Control Limits by Product Code for more info.



Name

This is the required unique name of the control limit as it will appear in selection lists and control charts. It is better to keep this short in length so that it will fit better on the control charts.



Kind

Each type of control chart has control limit kinds that it works with. If a control limit will be used with an Individual control chart, then either the Individual LCL (Lower Control Limit), Individual UCL (Upper Control Limit) must be used.

> **Note**
>
> Each Control Limit kind has **Other** as an option. This kind is there for legacy support and has deprecated. Do not use this kind.

Calculation Script

The SPC module uses python scripts to calculate control limits. This allows the user to override the default calculation of a control limit or add new control limits that are not provided by default. Additionally, they can be removed, cleaning up selection lists of control limits that may never be used.

When a user or script function is used to initiate a control limit to be calculated, the script in the associated control limit is executed. An event object is passed into the script that contains the information and data used to calculate the new control limit value. See Control Limit Event object for more information.

In the example, any lines that start with the pound (#) character are comments and are ignored when the script is executed.

The event.getData() on line 8, returns the samples that will be used to calculate the new control limit. It is a data set (see Ignition Data set in scripting for more information) and contains a row of data for each sample. Each sample row includes measurement values, calculated values (such as xBar, standard deviation, etc), sample date and time. For the p and u charts where the control limits can vary by sample, this data set includes columns to which the the newly calculated control limit for each sample can be saved.

The ds.getColumnIndex on lines 11 and 12, returns the column number of the XBar and Range columns. This is done for speed reasons because it is faster to reference the column by number instead of finding the column by name.

From line 19 to 21, each sample row in the data set is cycled through. This is done to total the xBar and range values. The ds.getValueAt() function returns the value in the data set for the specified row and column.

Line 24 calculates the average of the xBar values, also known as x double bar (XDBar).

Line 25 calculates the average of the range values, also known as range bar (RBar).

The event.getSampleSize() in line 28, returns the number of measurements per sample. This will be used to determine which a2 value to use from the array in line 5. The a2 is a factor to calculate the 3 sigma or 3 times standard deviation value and changes based on the number of measurements in each sample.

Lines 31 through 34 lookup the a2 value that is going to be used to calculate the new control limit value. A quick range check is done to prevent reading a value that is outside of the array limits.

Line 37 calculates the new UCL value.

And finally, the value is saved to pass back the new control limit value in line 40.

Default XBar UCL control limit calculation script

```
#XBar UCL Calculation
#Define the A2 factors array.
#The A2 factors correspond to the sample size which starts at 2.
#This is why element 0 and 1 of the array are 0.
a2 = [0.0, 0.0, 1.880, 1.023, 0.729, 0.577, 0.483, 0.419, 0.373, 0
.337, 0.308, 0.285, 0.266, 0.249, 0.235, 0.223, 0.212, 0.203, 0.19
4, 0.187, 0.180, 0.173, 0.167, 0.162, 0.157, 0.153]

#Get the SPC data that the XBar UCL will be calculated for
ds = event.getData()

#Get the columnn indexes within the SPC data
xBarColNdx = ds.getColumnIndex("XBar")
rangeColNdx = ds.getColumnIndex("Range")

#Initialize xBar and range sums that are need to calculate
average xBar and range.
xBarSum = 0.0
rSum = 0.0

#Cycle through each row and add to the sums
for row in range(ds.rowCount):
    xBarSum = xBarSum + ds.getValueAt(row, xBarColNdx)
    rSum = rSum + ds.getValueAt(row, rangeColNdx)

#Calculate the average xBar and range
xDBar = xBarSum / ds.rowCount
rBar = rSum / ds.rowCount

#Get the sample size.
sampleSize = event.getSampleSize()

#Lookup the A2 value
if sampleSize < len(a2):
  a2Value = a2[sampleSize]
else:
    a2Value = a2[len(a2) - 1]

#Calculate the xBar UCL
ucl = xDBar + a2Value * rBar

#Return the new xBar UCL back to the SPC module
event.setControlLimitValue(ucl)
```

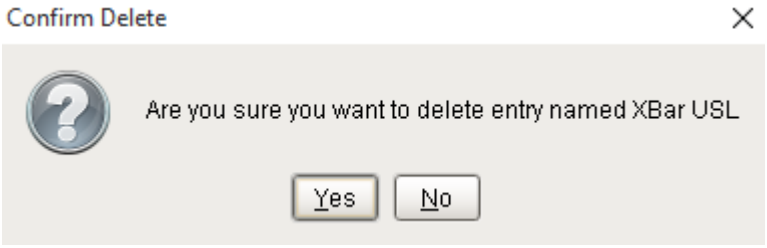
Group

A group name can optionally be added to each Control Limit. Control Limits that share the same group name will then have their value set automatically when any control limit in that group is set.
Example

An np Chart and a p Chart both show UCL spec limits. The np UCL and the p UCL Control limits have the same Group name - PRODUCT_UCL_GRP. When a change is made to the UCL limit on the np chart, the UCL value for the p UCL will also be changed

Deleting a Control Limit

To delete a control limit, right-click the Control Limits table on the Quality tab at the Enterprise level of the Production Model and select **Delete** from the drop-down menu. A window will appear confirming that you permanently want to delete the control limit.



Importing and Exporting Control Limits

Export

To export control limit entries, right-click anywhere on the table containing control limit entries and select the Export menu item. A dialog box will appear to allow selection of an existing file or the entry of a name for the new file to save the control limits to. If a file extension is not entered, then the default .csv will be used.

The first line of the file must contain at least the property names separated by commas. If additional names exist, they will be ignored. The property names can be in any order. Below is a sample csv file showing multiple control limit entries. The lines in the example shown below have been shortened.

Import

To import downtime entries, right-click anywhere on the control limit table and select the Import menu item. A dialog box as shown below will appear to allow selection of a comma separated values (csv) formatted file.

```
Name,Kind,Script
"c LCL","23","#c LCL Calculation\nimport math\n\n#Get the
"c UCL","22","#c UCL Calculation\nimport math\n\n#Get the
"Histogram LCL","29",""
"Histogram UCL","28",""
"Individual LCL","11","#Individual LCL Calculation\n#Get t
"Individual UCL","10","#Individual UCL Calculation\n#Get t
"Median LCL","14","#XBar LCL Calculation\n#Define the A2 f
"Median UCL","13","#XBar UCL Calculation\n#Define the A2 f
"MR LCL","35","#Moving Range LCL Calculation\n#The LCL for
"MR UCL","34","#Moving Range UCL Calculation\n#Get the SPC
"np LCL","20","#np LCL Calculation\nimport system\nimport
"np UCL","19","#np UCL Calculation\nimport math\n\n#Get th
"p LCL","17","#p LCL Calculation\nimport system\nimport ma
"p UCL","16","#p UCL Calculation\nimport math\n\n#Get the
"Range LCL","5","#Range UCL Calculation\n#Define the D3 fa
"Range UCL","4","#Range UCL Calculation\n#Define the D4 fa
"StdDev LCL","8","#Standard Deviation LCL Calculation\n#De
"StdDev UCL","7","#Standard Deviation UCL Calculation\n#De
"StdDev XBar LCL","2","#Standard Deviation XBar LCL Calcul
"StdDev XBar UCL","1","#Standard Deviation XBar UCL Calcul
"u LCL","26","#u LCL Calculation\nimport math\n\n#Get the
"u UCL","25","#u UCL Calculation\nimport math\n\n#Get the
"XBar LCL","2","#XBar LCL Calculation\n#Define the A2 fact
"XBar LSL","3",""
"XBar UCL","1","#XBar UCL Calculation\n#Define the A2 fact
"XBar USL","3",""
```

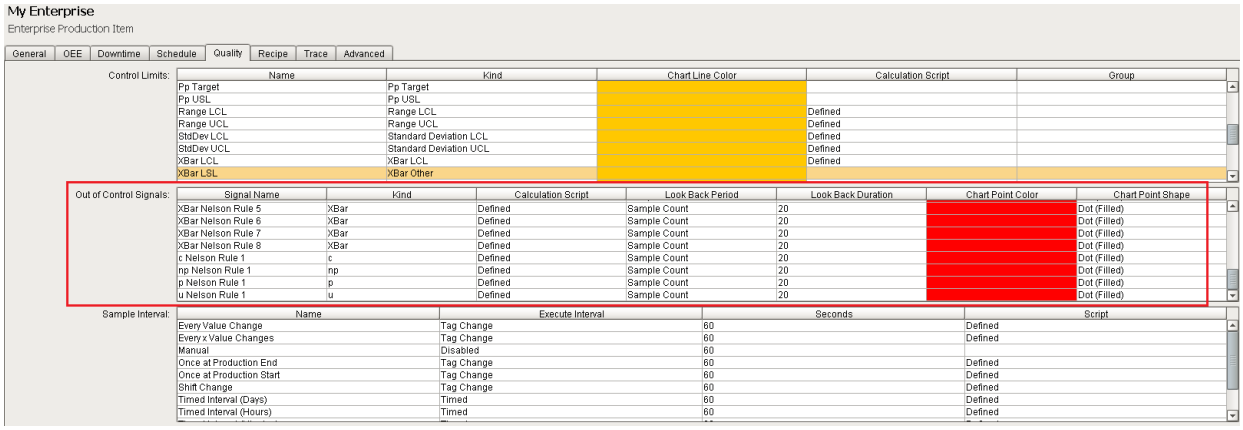
8.3.7 Out Of Control Signals

Out-of-control signals (rule violations) occur in a variety of situations, but all the signals indicate a change in the process where it is considered to be abnormal, or out of control. Some signals include: six points in a row that are increasing or decreasing, eight points in a row that are farther than one standard deviation away from the centerline, or fourteen points in a row that are alternating up and down. When used properly, these signals can identify important changes that can help to improve or maintain the process.

Signals can be configured so that they are evaluated every time new sample data is recorded. This allows for quick and automatic detection of out of control conditions. Once an out of control condition is automatically detected, Ignition provides a variety of actions that can be performed, such as standard alerting, communications, logging and more.

For automatic signal evaluation to be enabled, the Look Back Period must be set to something other than **No Auto Evaluation**, a valid look back duration must be set and the signal must be selected for the desired sample definitions.

Out of Control Signals can be added, edited or deleted on the Enterprise page in the designer under the **Quality** tab as shown .



Default Signals

When a new Enterprise Production Item is added, the following control limits are added:

- Individual Outside
- Out of Limits
- Outside Limits

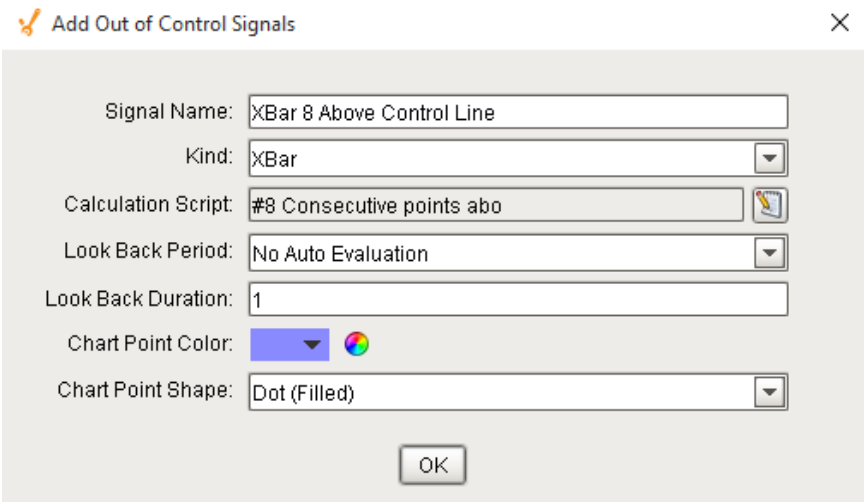
Adding and Editing Signals

To add an out of control signal, right-click the Out of Control Signals table and select **New** from the drop-down menu. A window will appear with several fields to be completed, including the signal name, kind, calculation script, lookback period, lookback duration, chart point color and chart point shape.

To edit an out of control signal, right-click the Out of Control Limits table and select Edit from the drop-down menu. A window as shown below will appear identical to the window used to add out of control limits. Once the desired fields have been edited, select OK.

Signal Name

This is the required unique name of the signal as it will appear in selection lists and control charts. It is better to keep this short in length so that it will fit better on the control charts.



Kind

Each type of control chart has signal kinds that it works with. If a signal will be used with a Individual control chart, then the Individual signal kind must be used.

Available control limits kind grouped by control chart type:

- XBar
- Range
- Histogram
- Individual
- MR
- Standard Deviation
- Median
- p
- np
- u
- c

Calculation Script

Because signal calculations can vary, the SPC module uses scripting. This allows the user to override the default calculation of a signal or adding new signals that the SPC module may not provide by default. Additionally, they can be removed, cleaning up selection lists of signals that may never be used.

Signals are evaluated when viewing them on control charts or when new sample data is recorded. When either of these trigger the signals to be calculated, the script in the associated signal is executed. An event object is passed into the script that contains the information and data to calculate the signal state values. We will introduce this event here but see Signal Evaluated Event object for more information.

Example-Description

In the example below, any lines that start with the pound (#) character are comments and are ignored when the script is executed.

Line 2 initializes a variable used to track how many consecutive calculated values (like the x bar value) are above the control line (like the x double bar value).

The event.getData() on line 5, returns the samples that will be used to calculate the signal state values. It is a data set (see Ignition DataSet in scripting for more information) and contains a row of data for each sample. Each sample row includes measurement values, calculated values (such as xBar, standard deviation, etc), sample date and time and control limits. There is also a column named the sample as the signal to save the signal state value. By setting the value of this column to a zero (0), the sample is in control for this signal, and by setting the value of this column to a one (1), the sample is out of control.

The ds.getColumnIndex on lines 8 through 10, returns the column number of the "XBar", "XDBar" and signal result columns. This is done for speed reasons because it is faster to reference the column by number instead of finding the column by name.

Starting with line 13, each sample row in the data set is cycled through.

Line 16 reads the calculated value that in this case is the xBar value.

Line 17 reads the average of the calculated values, which in this case is the xDBar value.

In line 20, a test is done for the xBar value being greater than the xDBar. If it is, further checking is done in lines 22 through 38. If it is not, then the consecutive count variable is reset and the signal state value is set to 0 for the sample in lines 42 and 43.

Line 22 adds to the consecutive count variable before checking if the threshold of 8 has been exceeded.

Line 25 checks if the consecutive count threshold has been exceed. If not, the signal state value for the sample is set to 0 and the consecutive count variable is left at its current value.

Line 28 checks if the consecutive count just exceeded the threshold. If it just did, the signal state values for the previous 8 samples are set to 1. This flags the current sample and the previous 7 samples as out of control.

The else statement in line 35 is a check that occurs if more than 8 consecutive xBar values exceed the xBar value. It sets the signal state value to 1 and leaves the consecutive count variable at its current value.

Default 8 consecutive points above control limit signal calculation script

```
#8 Consecutive points above control line signal calculation
consecutiveCount = 0

#Get the SPC data that the signal will be calculated for
ds = event.getData()

#Get the columnn indexes within the SPC data
xBarColNdx = ds.getColumnIndex("XBar")
xDBarColNdx = ds.getColumnIndex("XDBar")
resultColNdx = ds.getColumnIndex("XBar 8 Above Control Line")

#Cycle through each row and check signal
for row in range(ds.rowCount):

    #Get the values to compare
     xBar = ds.getValueAt(row, xBarColNdx)
    xDBar = ds.getValueAt(row, xDBarColNdx)

    #Test if the x bar value is above x double bar value
    if xBar > xDBar:
        #Add to the consecutive count
        consecutiveCount = consecutiveCount + 1

        #Test if less than 8 consecutive x bar values are above x
double bar
        if consecutiveCount < 8:
            #Write a zero to the result column, meaning we are in
control
            ds.setValueAt(row, resultColNdx, 0)
        elif consecutiveCount == 8:
            #Now 8 consecutive x bar values are above the x
double bar
            #Write a 1 into the last 8 row because, they are all
out of control
            ndx = row
            while ndx > 0 and ndx > row - 8:
                ds.setValueAt(ndx, resultColNdx, 1)
                ndx = ndx - 1
        else:
            #Over 8 consecutive x bar values are above x double
bar
            #Continue writing a 1 into the result because this
row is still out of control
            ds.setValueAt(row, resultColNdx, 1)
    else:
    #x bar value is below, reset the consecutive count
```

```
    #and write a zero to the result column, meaning we are in
control
    consecutiveCount = 0
    ds.setValueAt(row, resultColNdx, 0)
```

Look Back Period

This property defines the time units of the Look Back Duration property.

Period	Description
No Auto Evaluation	Disable automatic signal evaluation after new sample data is recorded.
Seconds	Time in seconds to display as look back period.
Minutes	Time in minutes to display as look back period.
Hours	Time in hours to display as look back period.
Days	Time in days to display as look back period.
Months	Time in months to display as look back period.

Look Back Duration

When automatic signal evaluation is used, this property, along with the Look Back Period property, defines the time range of samples to pass to the calculation script. The calculation script can then cycle through the range of samples to find out of control conditions.

Chart Point Color

For samples that are out of control, this is the color to display the sample value on the control charts.

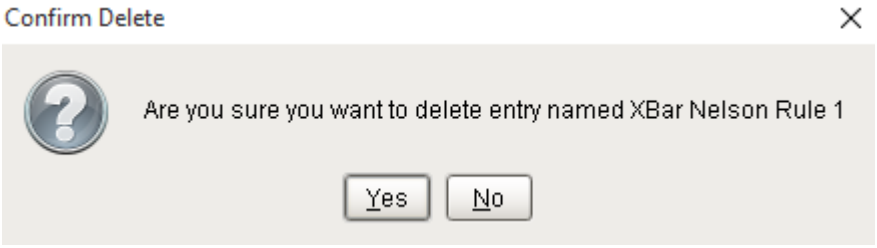
Chart Point Shape

For samples that are out of control, this is the shape to display for sample value on the control charts.

Looking at the default signal calculations along with the Scripting section of this manual and the Scripting section in the Ignition manual is the best method to learn all the possibilities of calculating signals.

Delete Signals

To delete an out of control signal, select the item to be deleted. After selecting, right-click the item and select Delete from the drop-down menu. A window as shown below will appear confirming that you permanently want to delete the out of control signal.



Importing and Exporting Signals

Export

To export signal entries, right-click anywhere on the table containing signal entries and select the Export menu item. A dialog box will appear to allow for the selection of an existing file or the entry of a name for the new file to which the out of control signal entries are saved. If a file extension is not entered, then the default .csv will be used.

The first line of the file must at least contain the property names separated by commas. If additional names exist, they will be ignored. The property names can be in any order. Below is a sample csv file showing multiple signal entries. The lines in the example shown below have been shortened.

Import

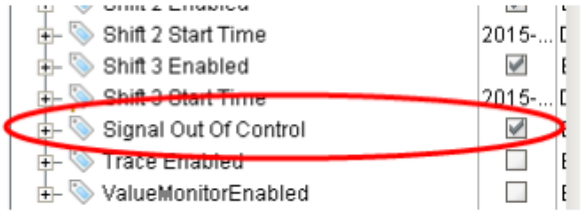
To import signal entries, right-click anywhere on signal entries and select the Import menu item. A dialog box will appear as shown below to allow selection of a comma separated values (csv) formatted file.

```
SignalName,SignalKind,SignalScript,SignalAutoEvaluatePeriod,SignalAutoEvaluateDuration,SignalChartColor,SignalChartShape
"Individual Outside","5","ds = event.getData()\nxBarColNdx = ds.getColumnIndex(\"XBar\")\nuclColNdx = ds.getColumnIndex(\'
"Out of Limits","1","ds = event.getData()\nrangeColNdx = ds.getColumnIndex(\"XBar\")\nuclColNdx = ds.getColumnIndex(\"XBar
"Outside Limits","2","ds = event.getData()\nrangeColNdx = ds.getColumnIndex(\"Range\")\nuclColNdx = ds.getColumnIndex(\"Ra
"XBar 8 Above Control Line","1","#8 Consecutive points above control line signal calculation\nconsecutiveCount = 0\n\n#Get
"XBar 8 Below Control Line","1","#8 Consecutive points below control line signal calculation\nconsecutiveCount = 0\n\n#Get
```

Rule Monitoring

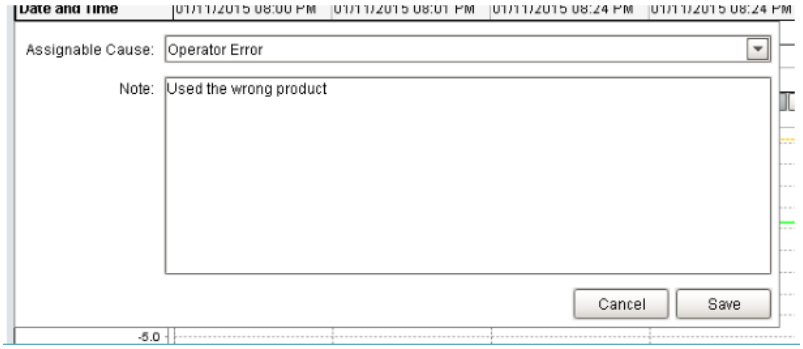
Typically, SPC software requires that someone opens a screen and visually checks for out of control conditions. Just like the scheduling of samples, someone may be distracted by other pressing production issues and fail to complete the task. The Sepasoft SPC module has powerful features that will automatically evaluate out of control signals every time new sample data is recorded. This can be expanded to instantly inform all parties that should know of various out of control conditions. Any sample that goes out of control will cause the **Signal Out of Control** tag to go to true and will show up in the control chart. Depending on the rule you will see a colored shape on every point that is out of control.

The SPC module provides an **Signal Out of Control** tag that goes to true when an out of control condition is detected.

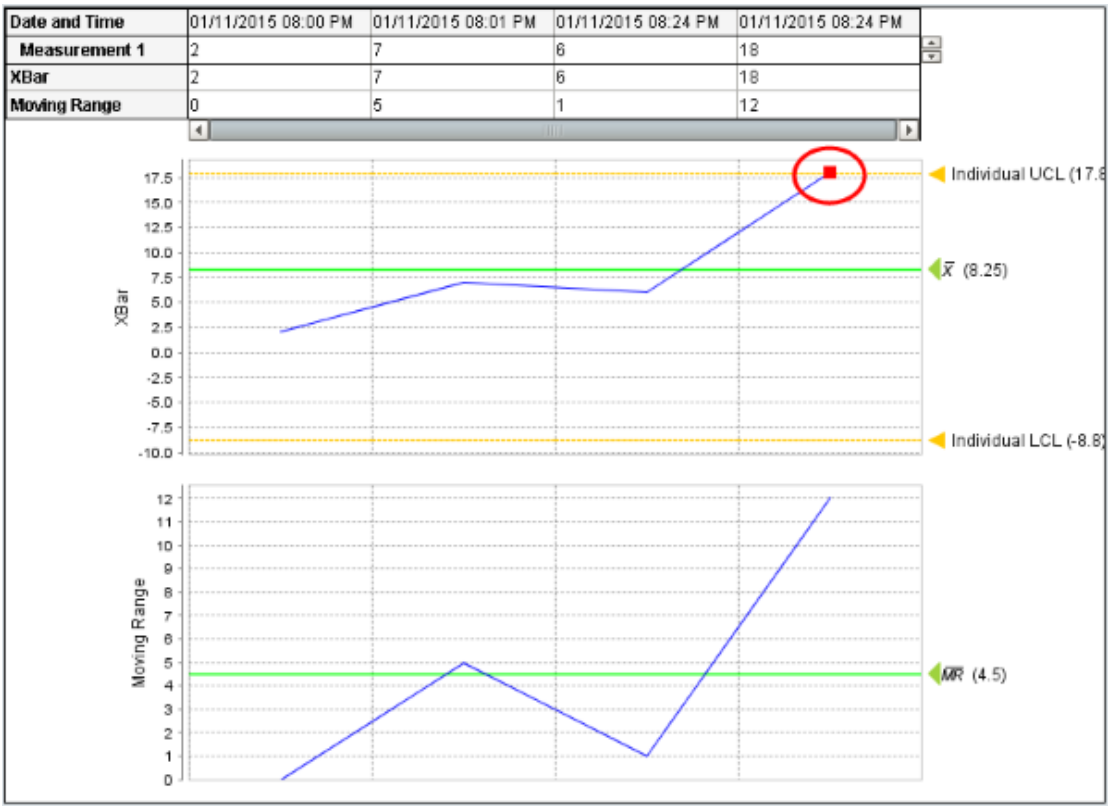


You can configure an alarm on this tag to notify people when the sample goes out of control.

If you pull up the control chart, you will see an indication of the out of control sample. You have the ability to right click on the point and assign a cause and enter in a note.



Keep in mind that you only get one set of tags per location which will show the signal out of control, if any of the sample definitions are out of control.



8.3.8 Sample Intervals

Samples can always be taken manually, but the SPC module also supports the automated scheduling of samples.

Sample Intervals are used to define the amount of time or number of readings that pass between samples. For example, the interval may be a timed interval that occurs every three minutes, every 100 readings, or samples can be taken continuously. These options will be available when defining a sample on the Sample Definition page when adding or editing a location. The intervals can also also be defined when creating Tag Sample Collectors in the Production Model.

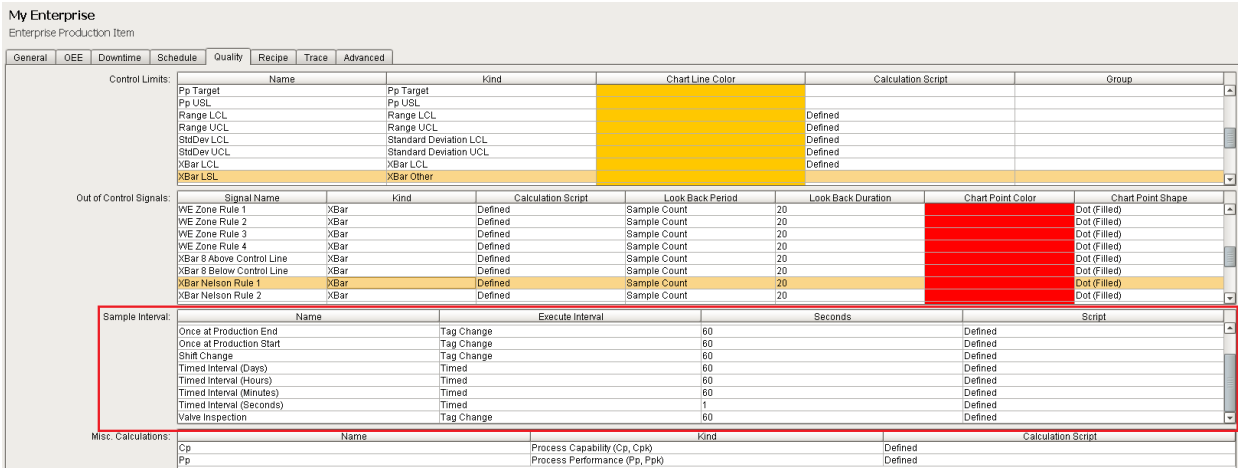
Sample Intervals can be added, edited or deleted on the Enterprise page of the designer under the **Quality** tab as shown.

Interval Types

When a new Enterprise Production Item is added, the following default intervals are added:

- Every Value Change
- Every x Value Changes
- Manual
- Once at Production End

- Once at Production Start
- Shift Change
- Timed Interval (Days)
- Timed Interval (Hours)
- Timed Interval (Minutes)
- Timed Interval (Seconds)
- Value Inspection



Adding and Editing Intervals

To add a sample interval, right-click the Sample Intervals table and select **New** from the drop-down menu. A window as shown will appear with several fields to be completed, including the name of the sample interval, as well as the scripting necessary to use the sample interval.

To edit a sample interval, right-click the Sample Intervals table and select **Edit** from the drop-down menu. A window will appear identical to the window used to add sample intervals. Once the desired fields have been edited, select **OK**.

Name

This is the required unique name of the interval as it will appear in selection lists.

Execute Interval

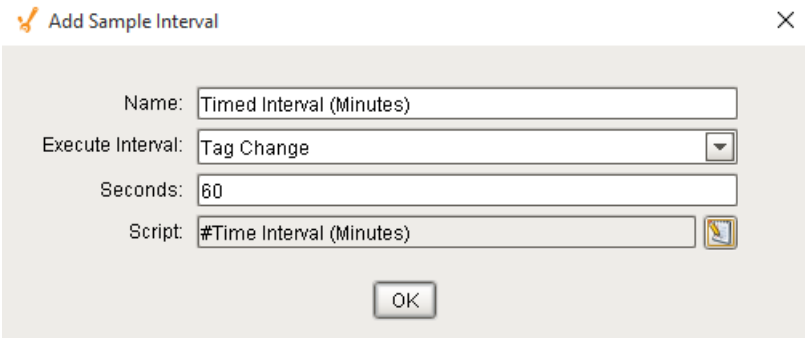
The **options** are:

- Disabled
- Tag Change
- Timed

Seconds

If the **Execute Interval** is set to **Timed**, the sample interval will get executed every 60 seconds, if you set it to 60, for each location defined for a sample definition.

> ⚠ Although the Interval script will execute every *x* seconds, Production Model restarts, project saves, and restarting the Gateway will cause the Interval scripts to fire immediately. You must use logic to check if samples have already been scheduled, and this can be achieved by using the **event.getSecSinceLastSampleScheduled()** function call.



Script

Because the default intervals may not be exactly what you are looking for, the SPC module uses scripting. This allows the user to override the default calculation of an interval or adding new intervals that the SPC module may not provide by default. Additionally, they can be removed, cleaning up selection lists of intervals that may never be used.

In the sample definition, an interval can be selected and will define when new samples are scheduled. These scheduled samples require manual entry of measurements.

In the Tag Sample Collector configuration, an interval is used to define when to automatically add new samples.

Example

In the example script, any lines that start with the pound (#) character are comments and are ignored when the script is executed.

Line 2 will allow us to use the Calendar object to do math with date values. See the Ignition documentation for more information.

Line 5 returns the seconds since the last time a sample was scheduled. There is a wealth of information in the event object that can be used to determine if a sample should be scheduled or taken. See Sample Interval Event Object.

Line 8 returns the duration to use. In this case it is in minutes.

Line 9 returns the coming due minutes. It is going to be used to schedule a sample prior to the time it is due, so that it will show in the sample list component prior to the time it is actually due. For Automatic Tag Sample Collectors, the coming due will be 0 and the sample will be recorded and measurements collected when the sample is created.

Line 12 does the actual checks to determine if a new sample should be scheduled. If secSinceLastSample equals None, then it means a sample has not been scheduled for the sample definition and location that is being checked. In this case, a new sample should be created.

Lines 15 through 17 calculate the scheduled start time for the sample. This is the time that the sample will appear in the sample list component and set the Sample Coming Due tag associated with the production location.

Line 20 sets the create sample flag that tells the SPC module to create a new sample after executing this script. This can be done through script functions specifically for creating samples, but this simplifies the task of doing so down to one line of script.

Time Interval (Minutes) script

```
#Time Interval (Minutes)
from java.util import Calendar

#Get the last time a sample was scheduled
secSinceLastSample = event.getSecSinceLastSampleScheduled()

#Calculate the interval in seconds
intervalSec = event.getInterval() * 60
comingDueSeconds = event.getComingDueMin() * 60

#If a sample has not been scheduled or intervalSec has expired,
schedule a new sample
if secSinceLastSample == None or secSinceLastSample >=
intervalSec - comingDueSeconds:

    #Schedule next sample to start now + coming due minutes
    cal = Calendar.getInstance()
    cal.add(Calendar.SECOND, int(comingDueSeconds))
    event.setScheduleStart(cal.getTime())

    #Create new sample - no values are recorded
     event.setCreateSample(1)
```

Delete Intervals

To delete a sample interval, select the item to be deleted. After selecting, right-click the item and select **Delete** from the drop-down menu. A window will appear as shown confirming that you permanently want to delete the sample interval.

Importing and Exporting Intervals

To export interval entries, right-click anywhere on the table containing interval entries and select the Export menu item. A dialog box will appear to allow selection of an existing file or the entry of a name for the new file to which the interval entries are saved. If a file extension is not entered, then the default .csv will be used.

The first line of the file must at least contain the property names separated by commas. If additional names exist, they will be ignored. The property names can be in any order. Below is a sample csv file showing multiple interval entries. The lines in the example shown below have been shortened.

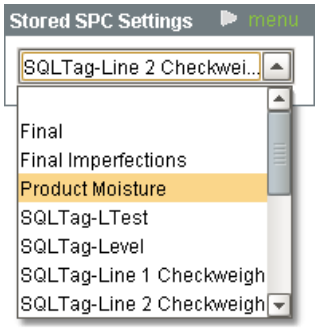
To import interval entries, right-click anywhere on interval entries and select the Import menu item. A dialog box will appear as shown below to allow selection of a comma separated values (csv) formatted file.

```
QualityIntervalName,QualityIntervalScript
"Every Value Change","#Record sample every time a tag value changes\nif event.isVal
"Every x Value Changes","#Every x value changes\nif event.isValueChangedEvent():\n\
"Manual",""
"Once at Production End","#Once at production enb \nif event.isTraceEndedEvent() ==
"Once at Production Start","#Once at production start \nif event.isTraceStartedEver
"Shift Change","#Once at shift change \nif event.isShiftChangeEvent() == 1:\n\tever
"Timed Interval (Days)","#Time Interval (Days)\nfrom java.util import Calendar\n\n#
"Timed Interval (Hours)","#Time Interval (Hours)\nfrom java.util import Calendar\n\
"Timed Interval (Minutes)","#Time Interval (Minutes)\nfrom java.util import Calenda
"Timed Interval (Seconds)","#Timed Interval (Seconds)\n#Test if product is being ru
```

8.3.9 Control Charts

Introduction To Control Charts

When a sample definition is created, it will appear as an option in the **Stored SPC Settings** selection box as shown below.

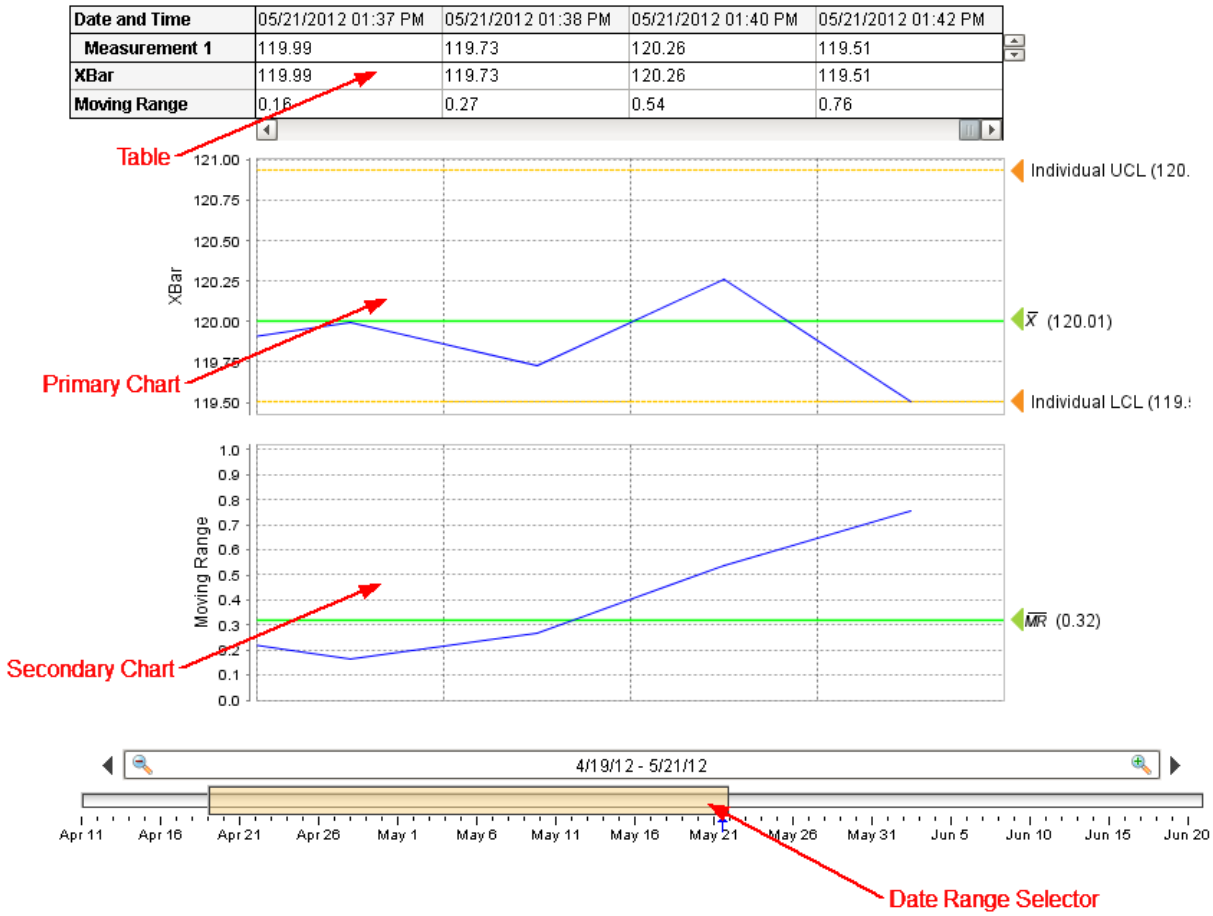


After selecting one of the Stored SPC Settings options, the control chart as defined in the Default Control Chart for the sample will be shown. The image to the side labels the major parts of a control chart. The Date Range Selector is used to select the date range of samples to view. It defaults to the current period of time, but can be used to select samples from the past. The table shows collected data and the calculated values. The calculated values that are included depends on the kind of control chart being displayed. When the scroll bar at the bottom of the table is moved to the left, the table, primary chart and secondary chart will all scroll in unison to previous samples within the selected date range.

For the attribute type of control charts the secondary table will not appear.

Changing which attribute is currently being shown in the control chart is done using the SPC settings panel. To change the attribute, click on the + select to the right of the Attribute label. This will show all of the attributes defined in the sample definition.

Control limits and signals can be selected or hidden using the same method as the attribute with the exception that more than one control limit or signal can be selected.

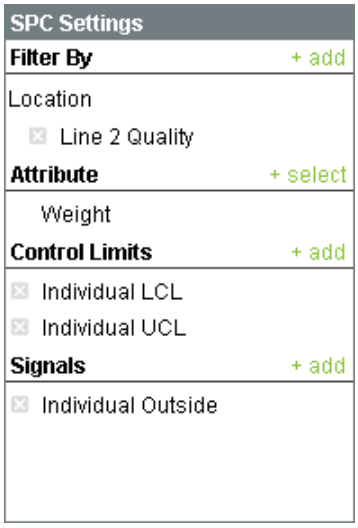


SPCGettingStartedChart

Control Chart

SPC Settings

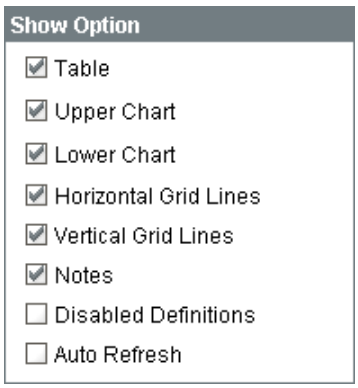
The filter by section allows for limiting the samples that will be shown and included in the calculated values. At a minimum, at least one location must be specified. This is because data collected from one location could be completely unrelated or in a different range than another location. If this is not the case, then multiple locations can be added to the filter.



SPC Settings

Show Option

The show options allow for the appearance of the control chart to be changed. By removing the Table option, the table will not appear leaving only the charts and allowing more samples to be viewed at once.



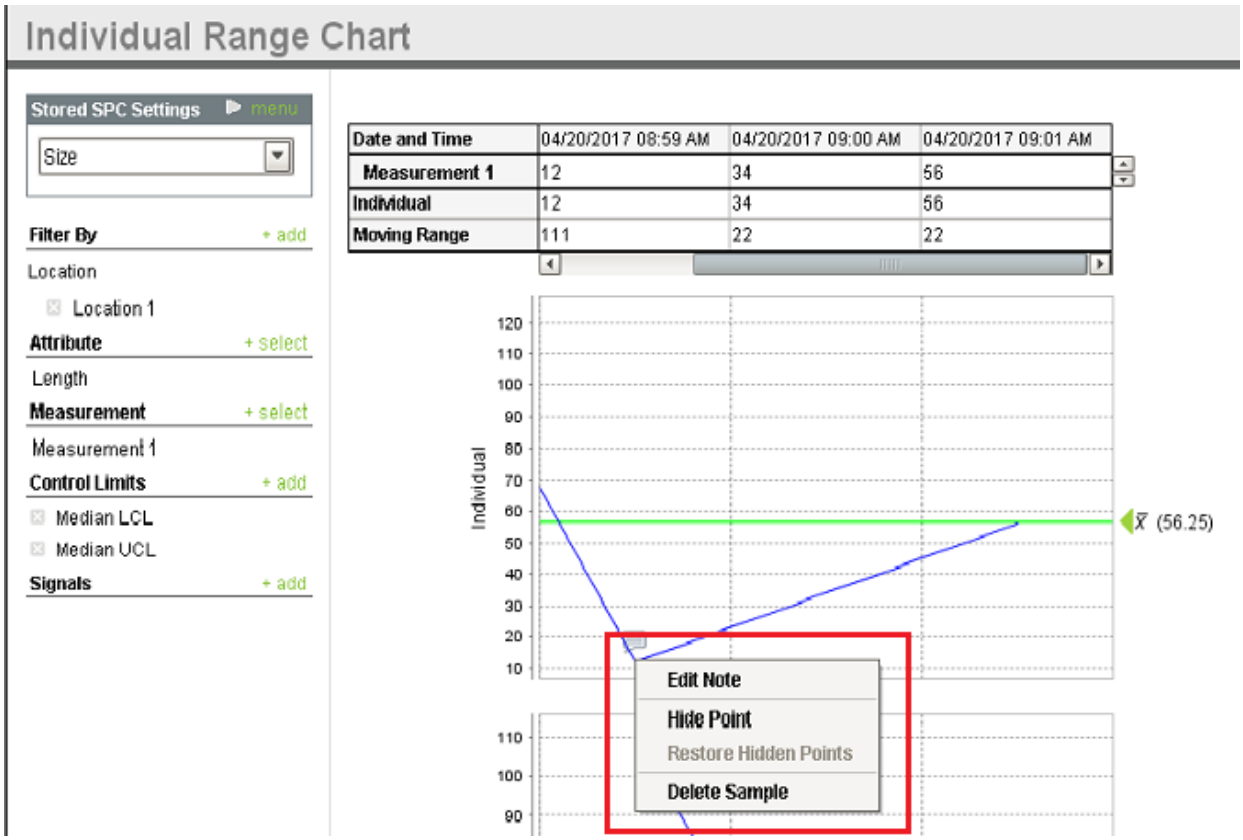
SPCGettingStartedShowOptions

Control Chart Show Options

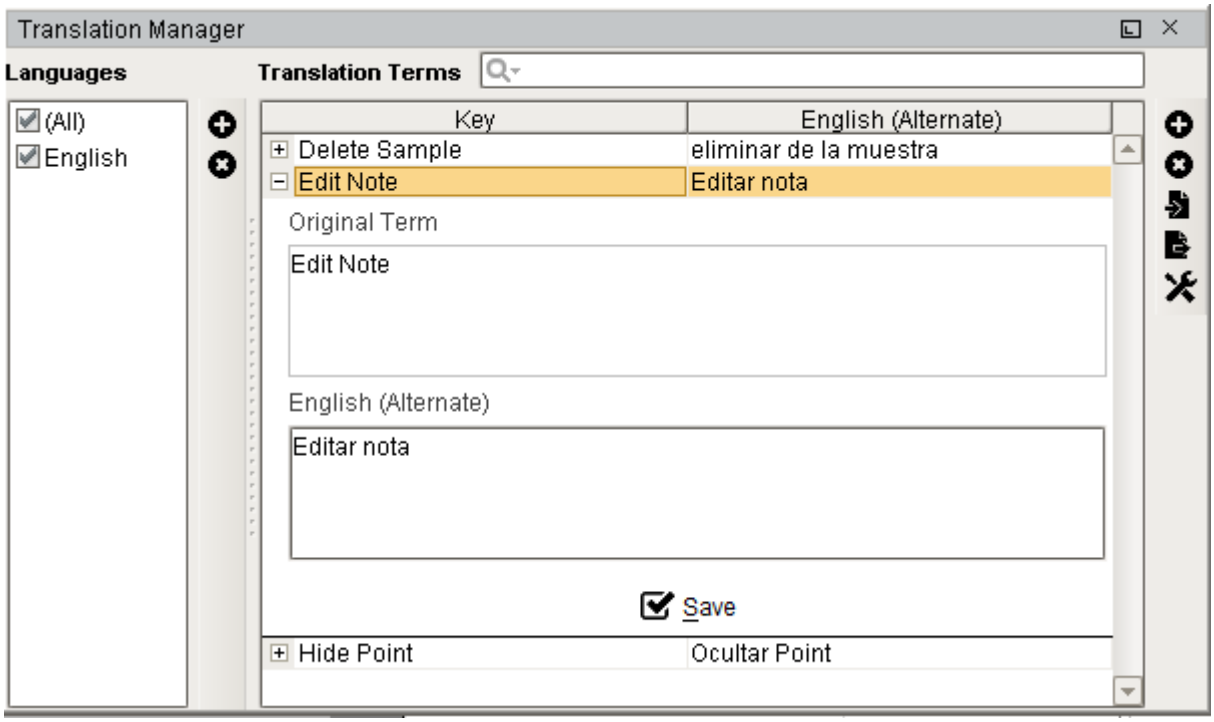
Control Chart Menu Items

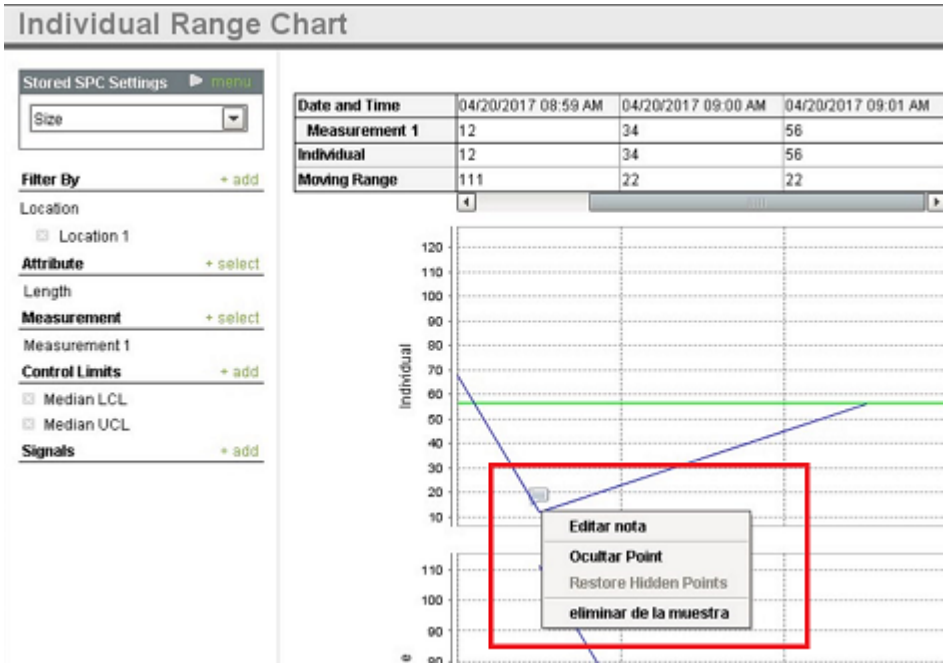
Right clicking the control chart will give the pop up menu items like Delete Sample, Edit Note, Hide Point and Restore Hidden Points as shown.

Localizable Menu Items



SPC chart popup panel menu strings are localizable. The alternate strings can be added through the Ignition translation manager (not the component translation manager). Reopen the control chart page and right click on a point to manifest the changes.





The control charts can be separated into three groups: **Value Charts**, **Attribute Charts**, and **Analysis Charts**. On all charts, it is possible to add assignable causes and notes to explain a data point.

Value Charts

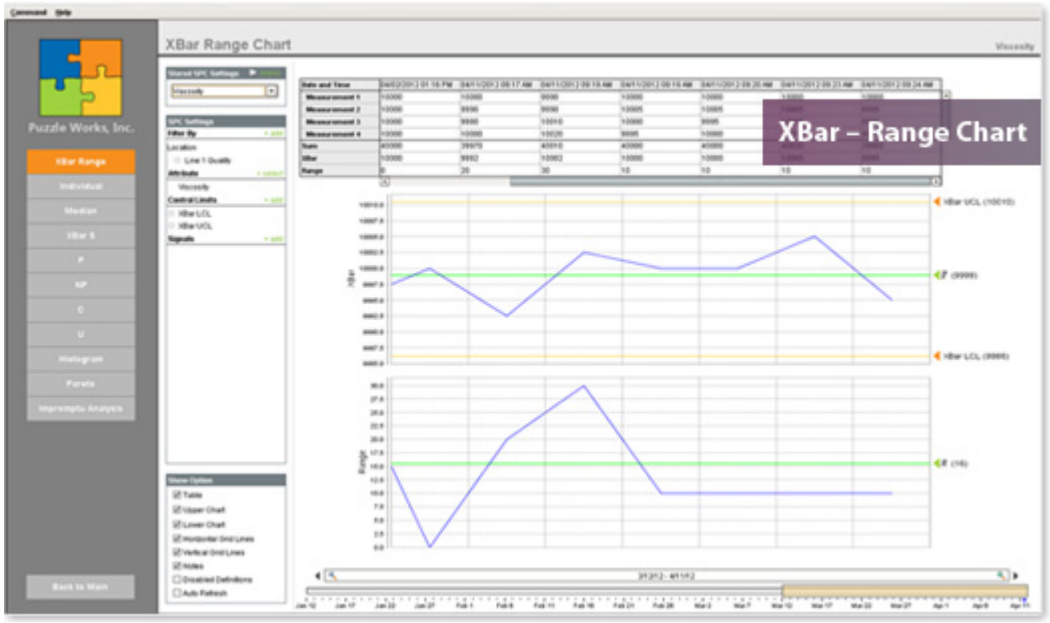
X Bar Range Chart

The X Bar Range chart (XBar and R) is used when there are multiple measurements taken in one sample and plots the process mean (Xbar chart) and process range (R chart) over time for variables data in subgroups, to help plot the stability of processes.

For example, if the pH is taken for five different pieces of product, the five different measurements will show up in the X Bar Range table. If all of these values are added together and then divided by the number of measurements taken, it will equal the average value, or x bar. This is what is graphed on the X Bar chart. When the lowest value is subtracted from the highest value, this equals the range, which is graphed on the Range chart. The range shows the overall consistency of the attribute being measured. The larger the range is, the less consistent the measurements are. If a point is not consistent with the rest of the data and is affecting other calculated values, this data point can be deleted. This will allow other calculated values, such as the x-double-bar and control limits, to reflect the data more accurately. X-double-bar is the average of all the averages, or the average of all the data points shown on the graph. Control limits are calculated to show where most data points on the graph will fall, provided the process is not out-of-control.

The X Bar Range chart should be used when data is generated frequently and is variable. This chart is useful for detecting small changes in the process and when multiple measurements are taken to represent a larger group of product.

Please see X Bar Range Chart component for more information.



X Bar Range Chart

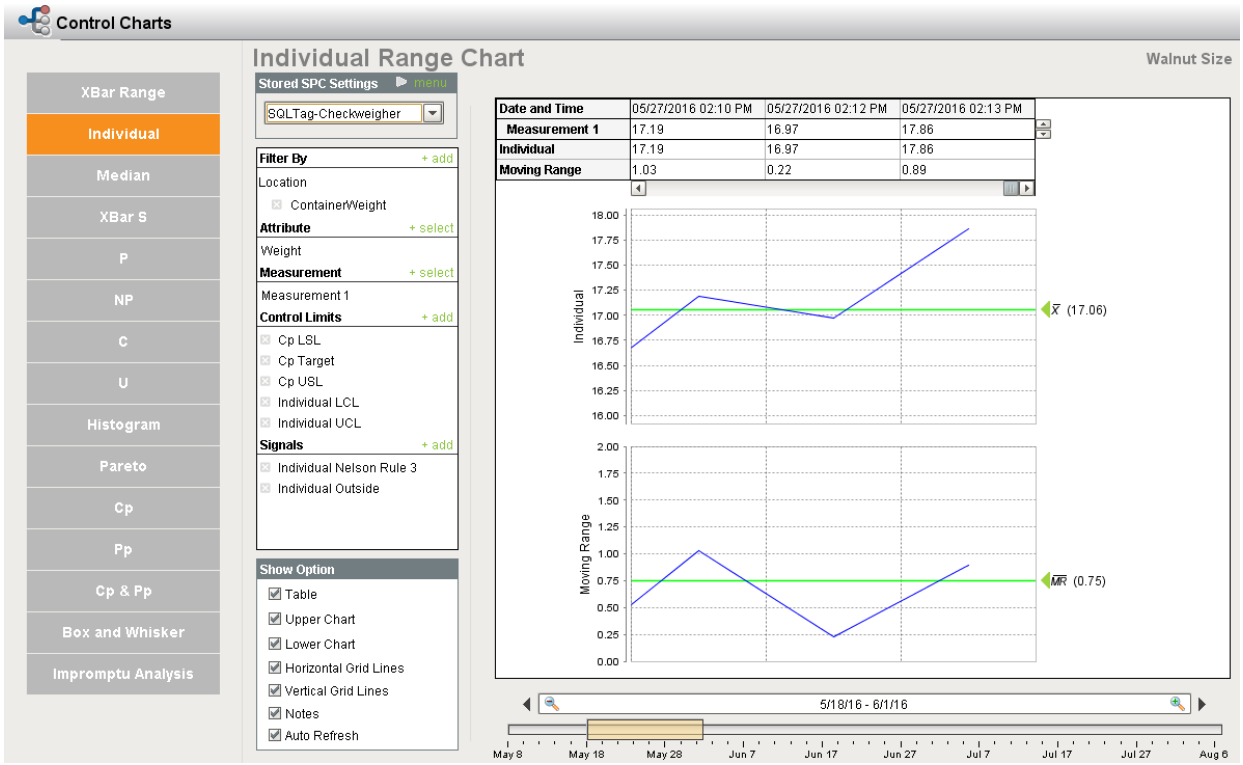
Individuals Chart

An individuals control chart (*XmR* chart, *I* -chart) can be used for time series tracking of a process to determine if the process is in statistical control and can be considered stable. When a process is considered stable, it experiences only common cause variability. When a process is not in control, special cause conditions can be causing nonstability.

The Individuals Chart is similar to the X Bar Range Chart, however, only one measurement is taken per sample instead of multiple. This means that the X Bar will always be the same value as the measurement, and a moving range will be calculated instead of the basic range. This means that instead of subtracting the lowest value from the highest value in one sample, moving range will calculate the difference between one sample and the next, showing the change from sample to sample. If a single measurement is used on the X Bar Range Chart, the range will always be zero, which fails to show the consistency between measurements.

Individuals charts are useful in situations when testing of a product results in the destruction of the product or if the testing is time consuming. It can also be used when a sample will yield the same result for a long period of time no matter how many measurements are made, such as batch operations. When using the Individuals Chart, the variable data should fall into a normal distribution, meaning the data points are equally likely to fall on either side of the average. This would appear as a bell curve on a histogram.

Please see Individual and Range Chart component for more information.



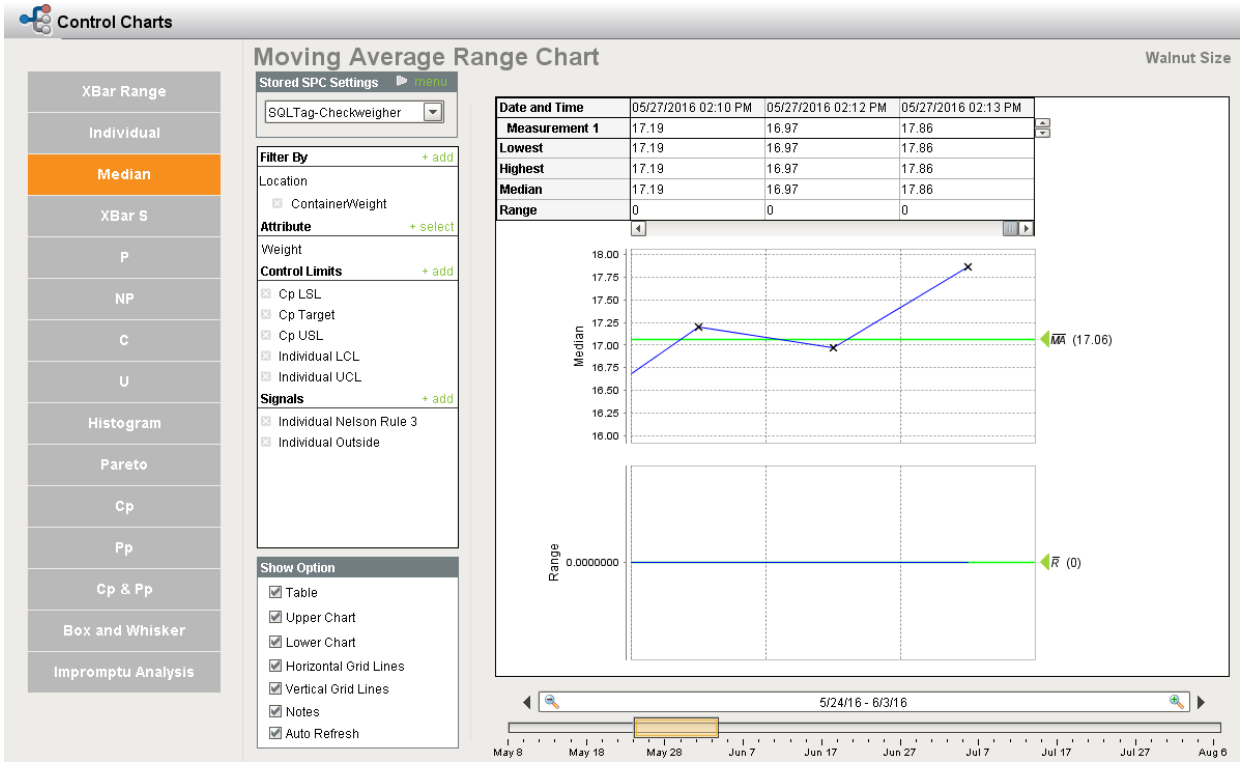
Median Chart

A median chart is a special purpose variation of the X-bar chart. This chart uses the median instead of the subgroup average to show the system's central location. The median is the middle point when data points are arranged from high to low. The chart shows all the individual readings and can be used to determine if the system is stable and predictable or to monitor the effects of process improvement theories. Although median charts show both central location and spread, they are often paired with range charts.

The Median Chart is also known as the MA-MR Chart or Moving Average-Moving Range Chart. Because data is generated slowly, the data on this chart is displayed differently. The first sample will contain three new data points. The second sample will contain the two most recent data points from sample one, in addition to one new data point. Sample three will contain the two most recent from sample two, as well as one new data point, and so on. Even though there are three samples with three data points each, there is only a total of five data points. On this chart, the median and the moving range are graphed. The median is the middle value based on the measurements in the sample (this is not the same as the average), while the range is the highest value minus the lowest value for each sample.

Like an individual chart, this chart should be used when the data is variable. In addition, data may also be costly or time-consuming to gather, or remain constant for a long periods of time. This chart should also be used when the data will not be normally distributed or when detecting small process changes.

Please see Median and Range Chart component for more information.

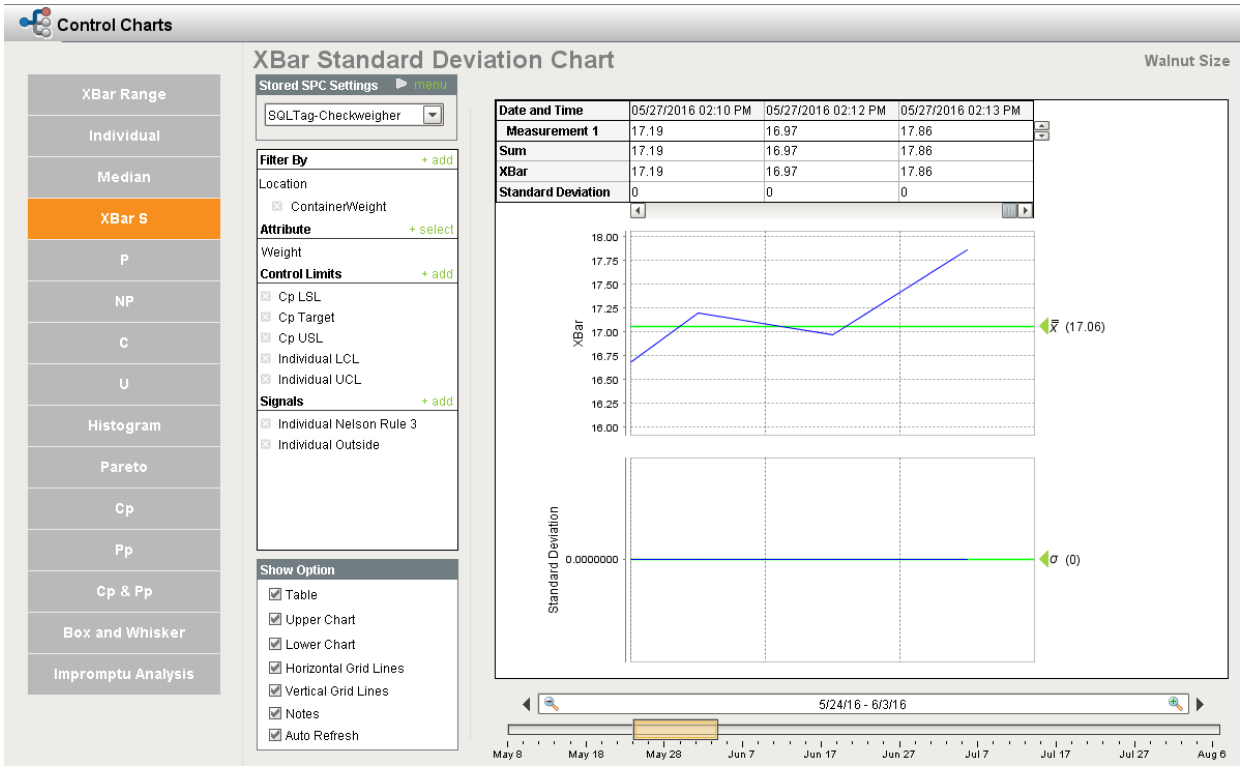


X Bar Standard Deviation Chart

An Xbar-S chart plots the process mean (Xbar chart) and process standard deviation (S chart) over time for variables data in subgroups . This combination control chart is widely used to examine the stability of processes in many industries.

This chart is very similar to the X Bar Range Chart. The major difference between the two is that the X Bar and S chart uses standard deviation to find the amount of variation within a sample instead of the range. Data must be in variable form to use this chart. It should also be used when data is plentiful enough that samples can have ten measurements or more, or when there is a need to rapidly detect small changes.

Please see X Bar and S Chart component for more information.



Process Capability

Capability or Process Capability refers to the statistical position of the normal distribution compared to the product or process specification. A process is capable when a bell curve is created by +/- 3 Standard Deviation and fits easily inside the desired specification. Indicators of capability are calculated based on the number of Sigma or Standard Deviations fitting between the process Mean and the closest specification.

- Cp or Cpi is the measurement of the ratio of Six Sigma divided into the allowable specification. Cpi does not indicate how well the process is performing, rather how good it could be.

Cp = Process Capability. A simple and straightforward indicator of process capability.
Cpk = Process Capability Index. Adjustment of Cp for the effect of non-centered distribution.

- Cpk is a typical indicator used to describe actual process capability. Cpk is used to determine the number of defects that are being produced, even if none have been found up to this point.
- Cpk is the capability on K side of the distribution. The K factor, or side, has the most risk and therefore is the worst of two possible measurements in a bilateral specification.
    - Cpk of 1.33 indicates 4 Sigma Capability or 4/3rds.
    - Cpk of 1.67 indicates 5 Sigma Capability or 5/3rds.

The greater the Cpk the less likely nonconformance will be present.

Ppk is an index similar to Cpk but considers more sources of variation in the process over a longer period of time.

Pp = Process Performance. A simple and straightforward indicator of process performance.
Ppk = Process Performance Index. Adjustment of Pp for the effect of non-centered distribution.

Capability is often misunderstood or considered a difficult concept. Most people are unaware that they are affected by Capability while driving to work. Here is an example of Capability in effect:

- Specification: width of one lane.
- Process: our car and its variation.
- Driving in our lane over a distance, the car easily stays within our allowable variation. We seldom hit the guardrail or oncoming traffic, indicating a very capable process.
- Cpk is likely to be 1.33 or greater.
- Larger vehicles have a smaller Cpk and smaller cars a larger Cpk.

Please see Process Capability Chart component for more information.

Attribute Charts

P Chart

P chart is a type of control chart used to monitor the proportion of nonconforming units in a sample, where the sample proportion nonconforming is defined as the ratio of the number of nonconforming units to the sample size, n. The number of nonconformities per item is irrelevant for this type of chart, which only tracks the total number of items; however, it is possible to have the types of nonconformities displayed on the same chart. P charts are used only when looking at the number of nonconforming items and when the sample size is not consistent.

Please see P Chart component for more information.

NP Chart

Unlike the P Chart, the NP chart requires that all the sample sizes are the same. The number of nonconforming items is graphed instead of the proportion because the samples can be directly compared. The types of nonconformities can also be displayed on the same chart. This chart should be used when counting nonconforming items when the sample size does not change.

Please see NP Chart component for more information.

C Chart

C - chart, also known as a count chart, is used to monitor **count** type data, typically total number of nonconformities per unit. It is also occasionally used to monitor the total number of events occurring in a given unit of time. Often, the types of nonconformities and their individual counts are noted as well. This chart is best used when counting nonconformities when the sample size will not vary. It is also important that each sample has equal opportunity for nonconformities.

Please see C Chart component for more information.

U Chart

U - chart is an attributes control chart used with data collected in subgroups of varying sizes. U - charts show how the process, measured by the number of nonconformities per item or group of items, changes over time. Nonconformities are defects or occurrences found in the sampled subgroup

Like the C Chart, the U Chart also graphs the number of nonconformities, but does so through a proportion. In this chart, the types and counts of nonconformities are tracked as well.This chart should be used when counting nonconformities when the sample size will vary. Also, if some samples have a greater opportunity for nonconformities than others, this chart should be used over the C Chart.

Please see U Chart component for more information.

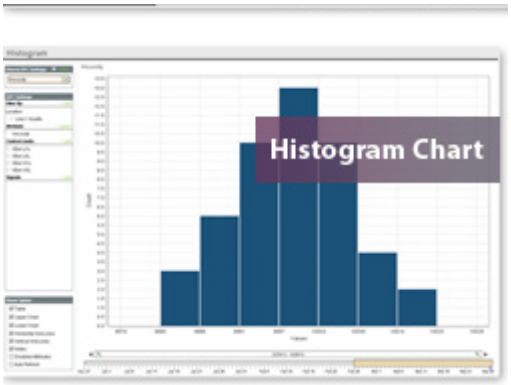
Analysis Charts

Histogram

A histogram shows the distribution of the data provided from the samples. A typical histogram has a **normal distribution**, meaning that most data points will fall in the middle of the graph and fewer will fall towards the outside, forming a bell curve. A distribution that is normal is just the most common pattern. There are other types of curves, such as skewed distribution or double-peaked distribution, which may be typical for certain processes. If a bell-shaped curve is formed on the histogram, then any variations in the data are most likely due to an assignable

cause. Assignable causes influence variations, which can occur in materials, environment, machines, peoples, etc. Ultimately, the histogram shows the consistency of a process. Histograms should be used when data is numerical and the shape of the distribution is to be observed. Observing the shape of the graph can help to determine whether or not the data is distributed normally, if a change has occurred in the process over time, or if two or more processes are different. This graph can also help to communicate with others about the data distribution or determine if a process will be able to meet the requirements of a customer.

Please see Histogram Chart component for more information.



Pareto

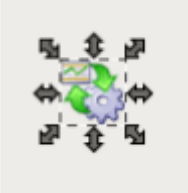
The Pareto chart , named after Vilfredo Pareto , is a type of chart that contains both bars and a line graph, where individual values are represented in descending order by bars, and the cumulative total is represented by the line.

The pareto chart is a bar chart that is used to show which factors are the biggest problems. The bars are arranged so that the most significant factor, that is, the factor that occurs the most frequently or cost the most(whether that be in time or money), is on the left, while the shortest bar, or least significant, is on the right. Because of the organization of the pareto chart, it is best used when looking at how often problems or causes occur and which of those are the most significant, or when looking at a specific component of a larger problem. Like the histogram chart, the pareto is also useful for the communication of data.

Please see Pareto Chart component for more information.

Building a Basic Control Chart

Let's look at building a simple control chart from scratch. You may want to put a control chart on an HMI window and you don't want all of the settings that come along with the built-in control charts window.
Now create a pareto chart for the Assembly sample definition. Open the Ignition designer and create a new main screen called **Simple Control Chart**. Drag the **SPC Controller** from the **SPC** tab in the component palette onto the window. It will look like this:



The SPC Controller is an invisible component that makes SPC data available for control charts, reports, and other components. The term invisible component means that this component appears during design time, but is not visible during runtime.

In cases where the SPC Selector offers too many options to the use, this component can be used. It has all of the same functionality as the SPC Selector but without the user interface. This means property bindings or script must be used to make the filter, compare by and data point selections. It also is used for providing data to canned reports and optionally allowing the user to make limited filter options. We need to set the properties to the appropriate values in order to show the pareto of the Assembly samples. Set the following properties:

Automatic Update = true

Auto Refresh = true (makes the data refresh automatically)

SPC Data Format = Pareto

Stored SPC Name =

Definition Name = Assembly

Attribute Name =

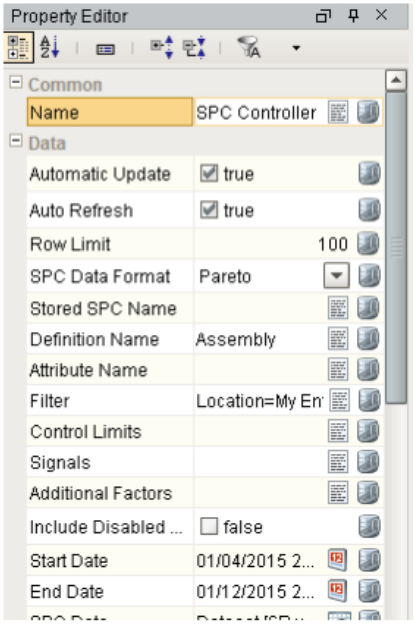
Filter = "Location=My Enterprise\Site 1\Packaging\Line 1\Line 1 Quality"

Control Limits =

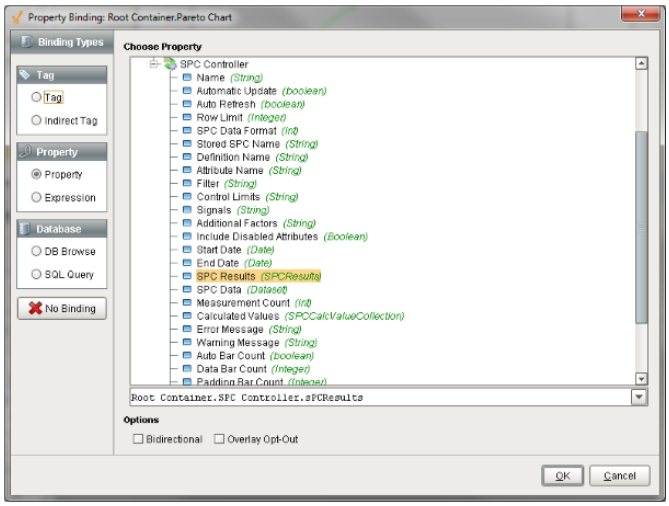
Signals =

Start Date = Appropriate start date

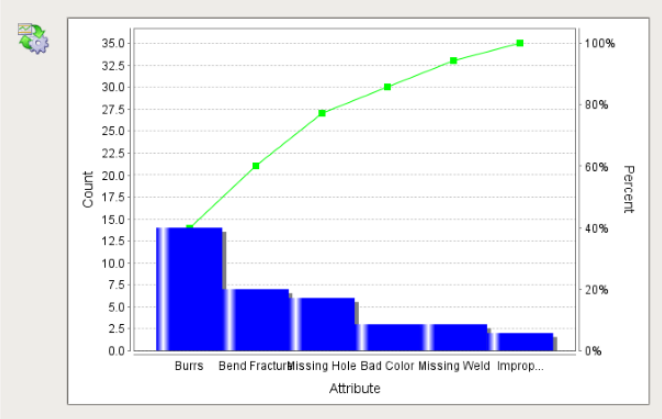
End Date = Appropriate end date



Make sure you have samples entered in and approved. Otherwise, you won't see any data in the control chart. Now that we have the data, let's add the control chart to the window. Drag on the **Pareto Chart** from the **SPC** tab of the Component Palette on the window. Bind the **SPC Results** property of the control chart to the **SPC Results** from the SPC Controller.



Press OK. You should see some data in the pareto.



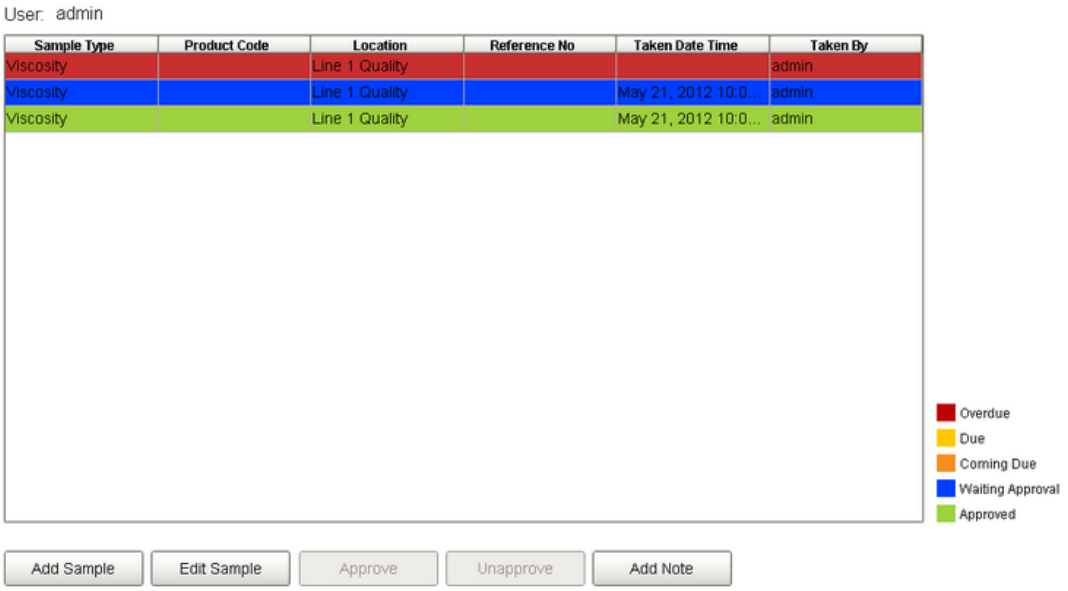
There are other settings you can configure on the control chart. With the auto refresh property set to true on the SPC controller, the pareto will continue to update as new data is added. You can perform the same process for any other sample definition and any control chart.

8.3.10 Scheduling and Entering Samples

Scheduling

Samples can be scheduled to be taken in the Sample Definition based on the Interval Type selected. The schedule sample list allows you to monitor the scheduled entries to see when they are coming due, due, overdue and waiting for approval and approved. Based on the color, users can easily see the current state of samples.

From this list, users can select a sample to enter measurements for or create new samples. See the sample definition section for more information about how to schedule samples or define them to be taken manually. By selecting a sample and clicking on the Edit Sample button, the sample data can be entered. Likewise, by clicking on the Add Sample button, a new sample can be added. Depending on the sample definition, samples can be automatically or manually approved. Once a sample has been approved, it will appear in the control charts and will be automatically evaluated for an out of control condition. In this demo, the Unapprove Sample button has been added to demonstrate the ability to correct previously approved sample data. This can be removed from the screen or allowed based on the user's security role.



SPCGettingStartedSampleList

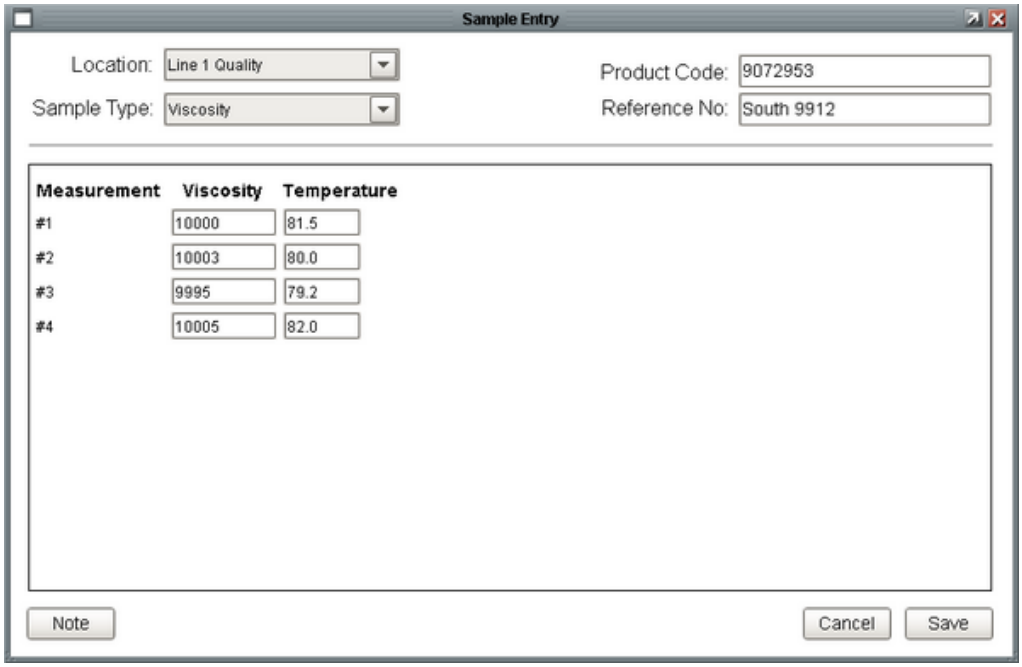
Sample List

Adding and Editing Samples

When a user clicks on the **Edit Sample** or **Add Sample** button, the sample entry form appears. If a new sample has been added, the location can be selected.

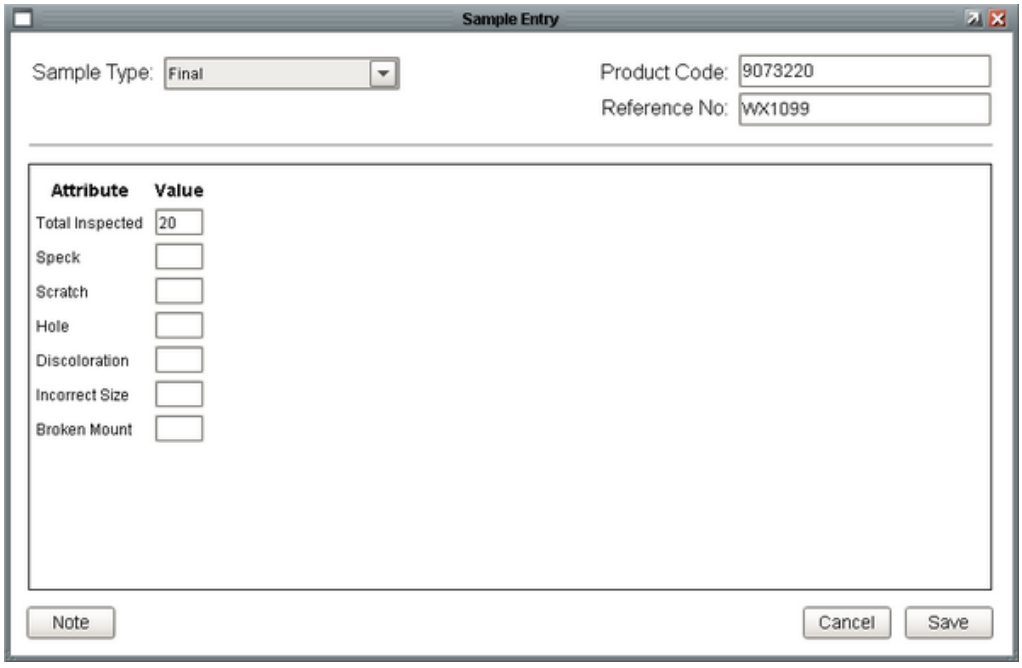
The following screen shows the entering of measurements for a value based sample. In this case, viscosity and temperature values. Users also have the ability to enter a product code and reference number (located in the upper right-hand corner). These can be used when viewing the samples in the control charts or for analysis beyond control charts.

If multiple measurements have been defined for each attribute, the attributes appear horizontally and the measurements vertically. If the sample definition only calls for one measurement, then the attributes will appear vertically.



SPCGettingStartedSampleEntry

Value Sample Entry



SPCGettingStartedSampleEntry2

Attribute Sample Entry

Approving Samples

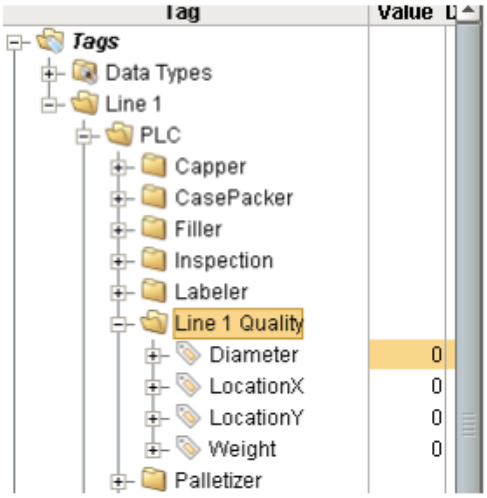
Samples can be configured in the Sample Definition screen to require the additional step of being approved or set to be auto approved. This provides the functionality of configuring an approval step with roles based security.

Sample values will not be shown in the control Charts until they have been approved.

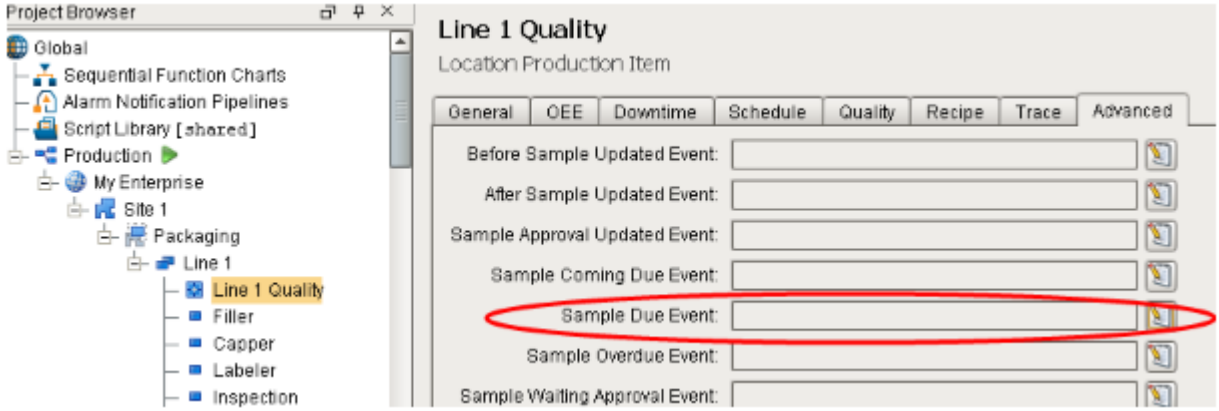
8.3.11 Automatic Multipoint Sample Entry

The tag collectors built into the SPC module only collect one measurement per sample. The primary reason for this is that once the sample is created, how to collect the measurements can vary greatly. For example, is it reading the same tag or a different tag, does it read consecutive values or delay between each measurement, etc.
This example walks you through how to automatically create samples with multiple measurements and attributes.
We already have a location and a sample definition to use. Let's use the Measurement sample definition for the **Line 1 Quality** location.
First, we need to create the tags that will be read in when entering the sample. In the Ignition designer, let's add 3 memory tags that will be the values for LocationX, LocationY, and Diameter.

Add the 3 memory tags to the **Line 1 Quality** folder in the **Line 1 > PLC** folder in the tag browser.
Set the names to the attribute names (LocationX, LocationY, Diameter) and set the datatypes to Float4. You should end up with the following:



Now we need to get the script in place that will take enter in the sample data. For that we need to select the **Line 1 Quality** location in the production model.
Select the **Advanced** tab on the right. We are going to configure a script on the **Sample Due Event** that gets fired anytime a sample is due.

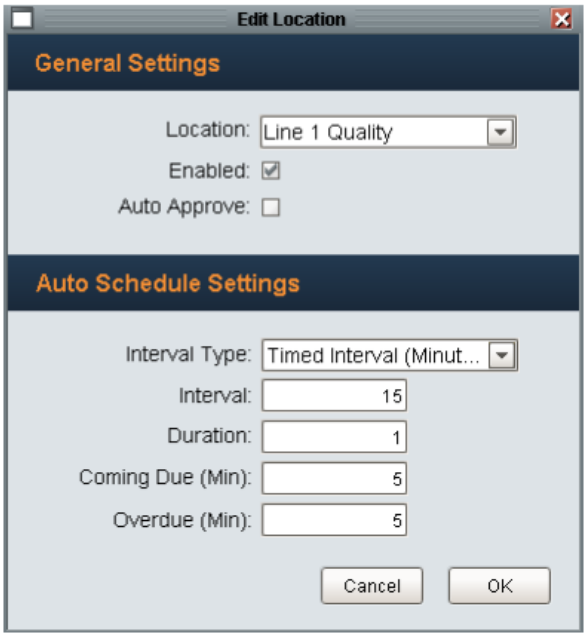


Click on the button to the right of the **Sample Due Event** and enter in the following script:

```
sample = event.getSample()
location = "[global]\My Enterprise\Site 1\Packaging\Line 1\Line 1
Quality"
sampleDef = sample.getSampleDefinition().getName()
if sampleDef == "Measurement":
    locationX = system.tag.read("[default]Line 1/PLC/Line 1
Quality/ LocationX").value
    locationY = system.tag.read("[default]Line 1/PLC/Line 1
Quality/ LocationY").value
    diameter = system.tag.read("[default]Line 1/PLC/Line 1
Quality/ Diameter").value
    sample.setSampleData(1, "LocationX", str(locationX))
    sample.setSampleData(1, "LocationY", str(locationY))
    sample.setSampleData(1, "Diameter", str(diameter))
    sample.setApproved(1)
    system.quality.sample.data.updateSample("QualityDemo",
location, sample, 1)
```

The script makes sure that the definition is **Measurement** before messing with it. That is because we have more than one definition for this location. You can see the script gets the values from the tags we created.
Save your changes in the designer.
Now that the script is in place we need to schedule the sample to be taken. We will use the scheduling settings on the location for the **Measurement** sample definition.
Navigate to the **Definition Management** window in the quality project runtime.
Select the **Measurement** sample definition. Right click on the **Line 1 Quality** location and

select edit.
Set the interval type to **Timed Interval (Minutes)**. Now the **Interval** setting applies. In this case it will specify how many minutes. Set the interval to 15 to schedule a sample every 15 minutes. Set the duration to 1, coming due to 5, and overdue to 5.



Press OK to save. Again press Save on the definition management screen to lock in the changes. You will see samples entered in every 15 minutes in a control chart.

8.3.12 Impromptu SPC Analysis

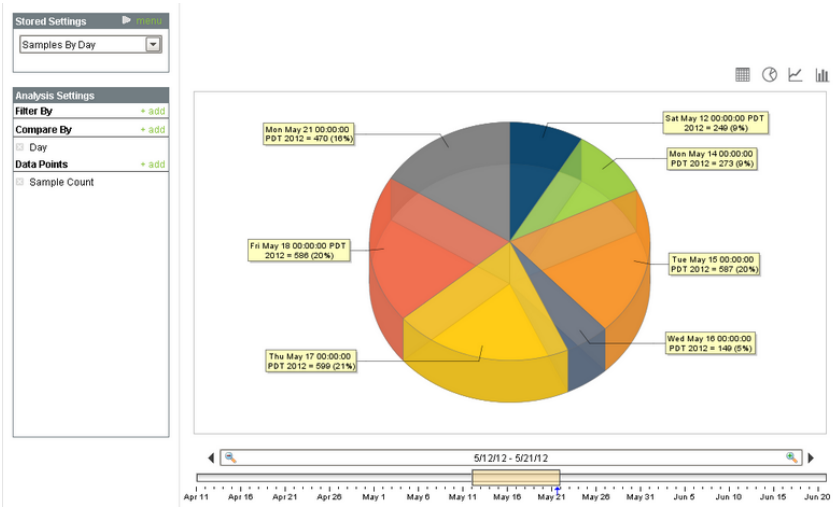
In addition to the Control Charts, the SPC module provides an impromptu SPC analysis screen allowing for free form analysis of production and quality data to help you zero in on the cause of quality related issues. This data can be filtered to include only specific criteria and comparisons can be made between different additional factors. For example, sample count by operator, or even process out of control conditions by operator and by shift, can be analyzed. You can select between the pie chart, bar chart, line chart, or tabular format for data analysis.

The date range selector at the bottom of the analysis screen is used to define the data range that will be included in the analysis. As you change the start or end dates, only the production runs that are within that range will be included in the analysis.

Stored Analysis

Start out by creating a new analysis by clicking on the menu of the Stored Settings panel and then selecting the New menu item. Next type in a name, select Quality for the type and click the OK button.

SPCGettingStartedAnalysisNew



SPC Analysis Screen

Filter By

Once an stored analysis has been created or selected, you can change the selections to zero in on the data that is desired. The filter section allows you to limit the data that is included in the analysis. Filters can be added by clicking on the + add icon on the right side of the Filter By section. Within the popup filter selection window, scroll down to the Shift option and click the icon. Notice the shifts can now be selected. Clicking on 1 for first shift will add the Shift = 1 causing the analysis results to included quality data for only for first shift. Any combination on filters can be added and the corresponding results will be shown.

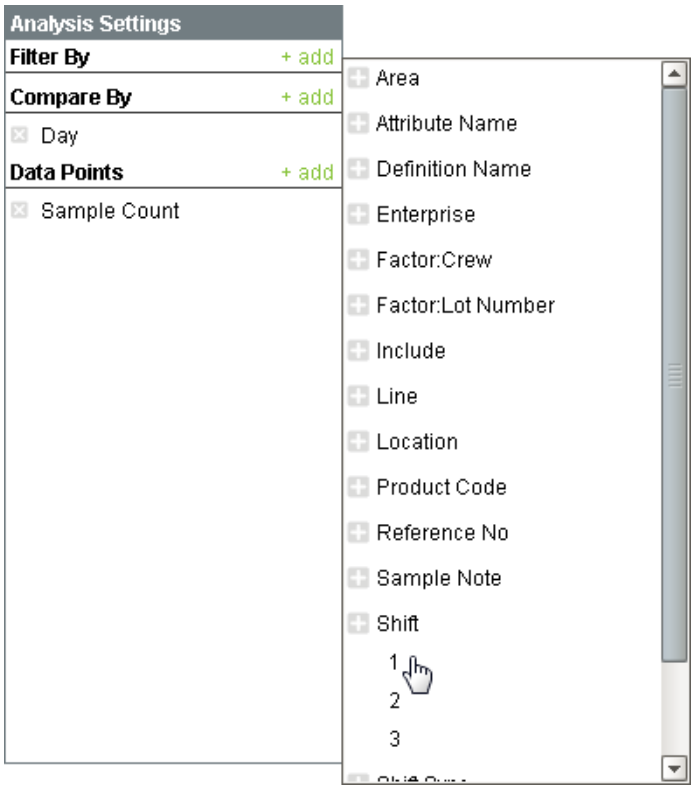
The list of available filters change based on the date range. For example, if no samples were taken during the second shift, then a 2 will not appear as an available option under shift. Filter By items can be removed by clicking on the located to the left of the filter name.

Compare By

Breaking up information into groups is more meaningful than just seeing a total for a given date range. For example, knowing the total sample count for a given data range does not provide actionable information that can be used to improve quality. Now, comparing by the sample count for each person entering sample data may provide meaningful and actionable data that can be used to determine staffing requirements.

Additional Compare By items can be added by clicking on the + add icon on the right side of the Compare By section. Within the popup Compare By selection window, click on the item that you want to compare analysis results between.

Compare By items can be removed by clicking on the ⊠ located to the left of the name.

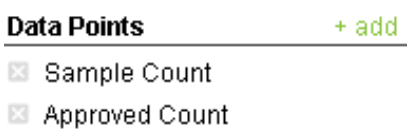


SPCGettingStartedAnalysisFilter

Filter By Options

Data Points

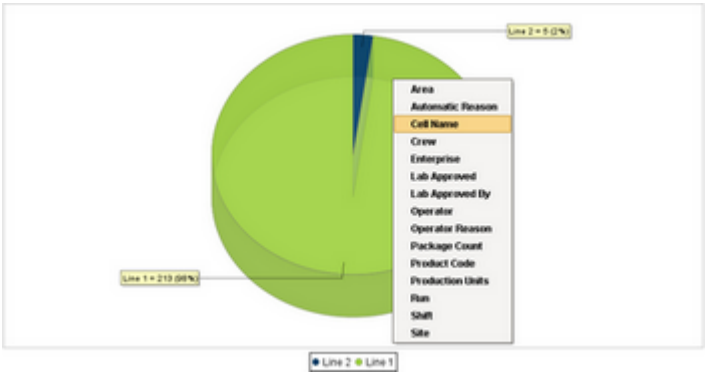
Data points are the individual pieces of information that will be present in the analysis. For example, sample count or approved count are just two of the many available data points. To add a data point, click on the + add icon on the right side of the Data Points section. Within the popup Data Point selection window, click on the data point item to include in the analysis.



Data Points can be removed by clicking on the ☒ located to the left of the name. The pie chart will only show one data point. For this reason if more than one data point is selected the bar chart, line chart or table must be selected to see all the selected data points.

Drill Down

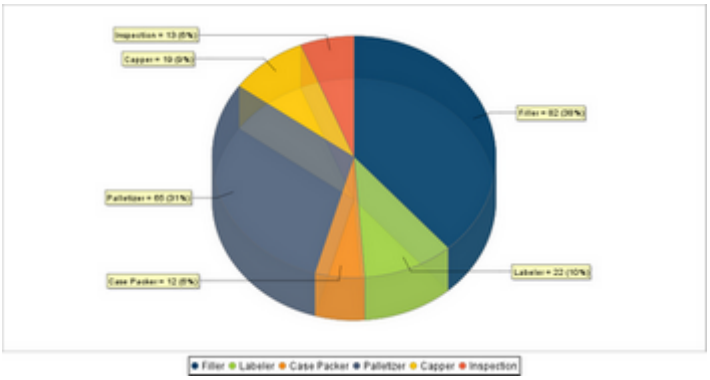
The drill down feature simplifies the compare by and filter selections. Click on a chart series to display the available drill down options. As shown in Drill Down Example 1 below, clicking on the Line 1 Quality pie segment will show a popup menu of drill down options. If the Shift option is selected, then the analysis filters will show the information by Shift and the Filter By and the Compare By sections add Shift. The result is shown in Drill Down Example 2. Again, by clicking on the pie segment and selecting another drill down option, the Filter By and Compare By selections will change to show the appropriate information. This can be continued any number of times.



Analysis Drill Down 1

Drill Down Example 1

Down Arrow



Analysis Drill Down 2

Drill Down Example 2

Using SPC Data in your Reports

The impromptu analysis allows you through the client window to access and aggregate data in a meaningful way. the same data can be accessed through scripting and through the SPC analysis controller component to return the required dataset to be passed to the Ignition report Designer to push to a third party reporting tool.

Quality Analysis Provider

Analysis providers determine which information will be viewed on a graph or pie chart. Based on which Analysis Provider is selected, some filter, compare by, and data point options may or may not be visible. This section covers the Quality Analysis Provider that is available with the SPC module.

The Quality Analysis Provider is used to query SPC information that is beyond what can be shown on control charts. For example, to determine the number of samples taken by user or the number of times a process was out of control over the last month cannot easily be shown in a control chart. This information can be combined with OEE, production, recipe monitoring and other information to create any kind of dashboard required.

Provider Name

Quality

SPCQualityAnalysisProvider

Filters

These are the filters that are available in the SPC Module. However, in addition to these filters, additional factors may be available if they are string data type. All additional factors start with **Factor:** For example, **Factor:Operator**. A filter will allow the user to see all of the data points in the analysis provider as it pertains to a specific area, shift, etc.

- Area
- Attribute Name
- Definition Name
- Enterprise

- Include
- Line
- Location
- Product Code
- Reference Number
- Sample Note
- Shift
- Shift Sync
- Site
- Tag

Compare By

These are the comparisons that are available in the SPC Module. However, in addition to these comparisons, additional factors may be available if they are string data type. All additional factors start with **Factor:** For example, **Factor:Operator**. A comparison allows one data point to be compared between all areas, days, etc.

- Approved By
- Area
- Attribute Name
- Day
- Definition Name
- Enterprise
- Line
- Location
- Month
- Note Entered By
- Product Code
- Reference Number
- Sample Entered At
- Sample Taken By
- Shift
- Site

- Tag
- Week
- Year

Data Points

These are the data points that are available in the SPC Module. However, in addition to these comparisons, additional factors may be available if they are string data type. All additional factors start with **Factor:** For example, **Factor:Operator**. Data points are the different values that will be presented or compared on a graph or chart.

- Approved At
- Approved By
- Approved Count
- Area
- Attribute Name
- Attribute Note
- Day
- Definition Name
- Enterprise
- Line
- Location
- Month
- Note Entered By
- Product Code
- Reference Number
- Sample Count
- Sample Entered At
- Sample Note
- Sample Taken At
- Sample Taken By
- Scheduled Finish
- Schedule Start

- Shift
- Site
- Tag
- Week
- Year

8.3.13 Analysing SPC Data in Control Charts

Viewing Sample Data in Control Charts

The impromptu control chart screen in the demo project allows you to view any sample definition. You can view the samples in different control charts and specify which attributes, signals, control limits, and filters you want to apply.

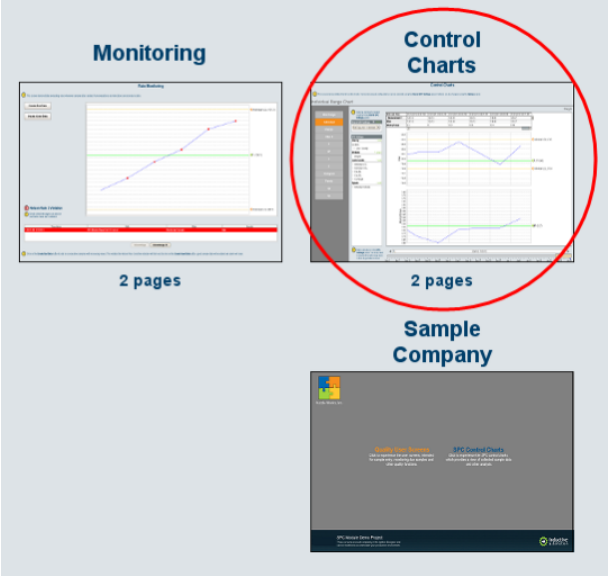
Analyze SPC Data with Customizable Charts

All the data in the world is of no use without good analysis tools, so we built the Sepasoft SPC Module with a full range of powerful and flexible SPC Control Charts. Based on security roles, control limit values can be calculated and set interactively on the Control Charts. The Additional Factors feature gives you the flexibility to associate and visualize other production information along with SPC data and with customizable appearance settings for charts, tables, control limits and signals, you have the power to see the information you need in the way you want.

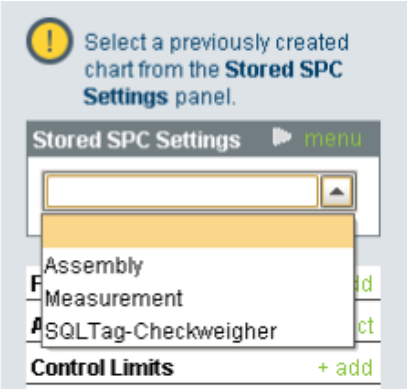
The control charts can be separated into three groups: value charts, attribute charts, and analysis charts. On all charts, it is possible to add assignable causes and notes to explain a data point. A sample note can be entered on the Lab or Test Stations page when the sample is first entered. This can be done by selecting a sample, then clicking Add Note. An attribute note is added directly from an SPC chart by right-clicking on a data point and selecting Set Note from the drop-down list. In addition to attribute notes, an assignable cause can also be added in this way. Assignable causes can also be saved for future use. Out-of-Control Signals and Control Limits can also be added to the graphs.

Now that we have collected some samples, we can start viewing them in control charts. Again there is a built-in screen in the quality demo project to view control charts.

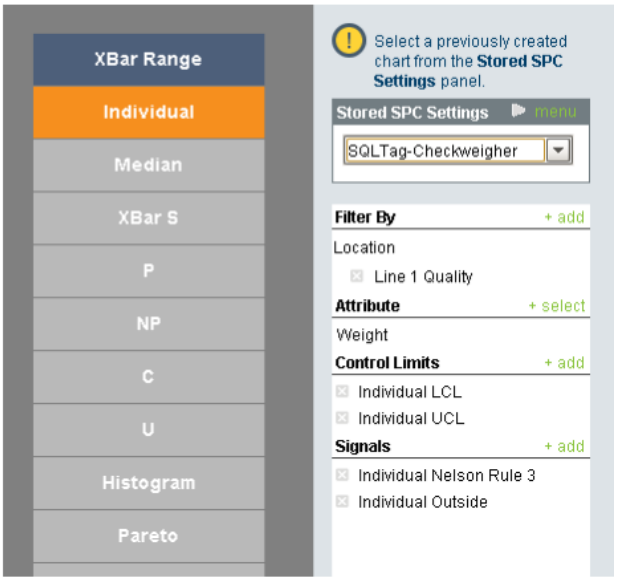
Open up the runtime for the quality demo project. Click on the Control Charts image.



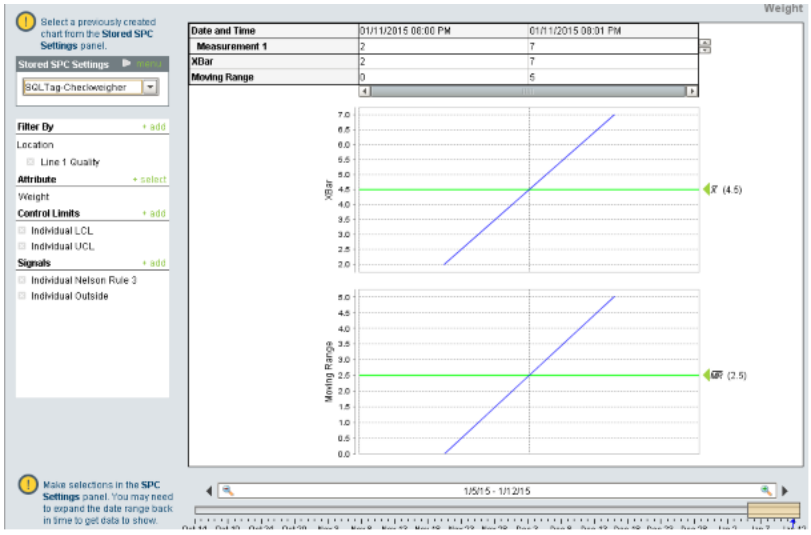
Here you can view any sample definition in any control chart (assuming the data works for that control chart). By default each sample definition automatically adds a Stored SPC Setting so you can easily view the control chart.
Click on the Stored SPC Settings dropdown to see the options.



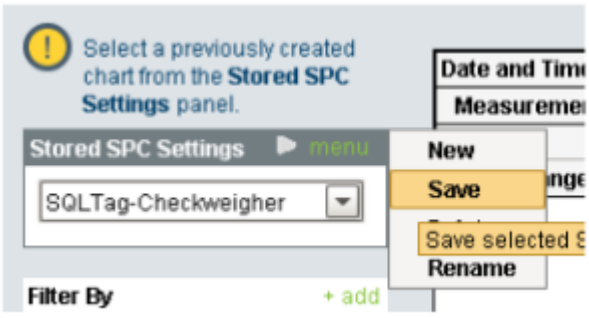
You can certainly create new Stored SPC Settings. Let's use the SQLTag-Checkweigher stored SPC setting. Once you select the setting make sure the Filter By is set to Line 1 Quality. The attribute should be set to Weight. If there was more than one attribute you can select which one you want to view. You can optionally select the limits and signals.
Lastly, select one of the control charts. Let's select the Individual chart for the checkweigher.



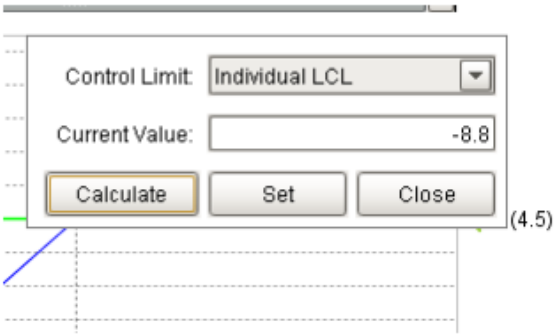
You may or may not have data. If you don't have any data go back into the designer and change the Weight tag 5 times for a sample to be entered in. Once you have done that come back to the control chart and change the start and end dates slightly to see the new data.



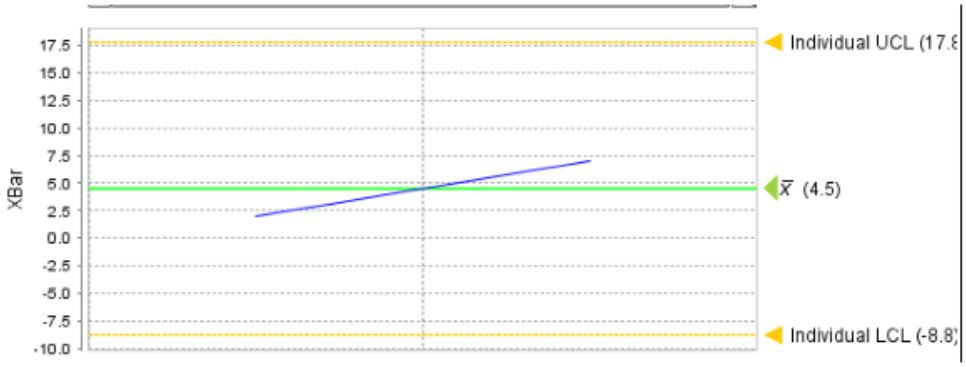
Save your Stored SPC Setting by clicking on menu and then save.



You can right click on the white space to the right of the control chart to set the LCL and UCL limits since they are selected. Select the limit and either set the value yourself or press calculate to let Ignition set it for you.



Now your chart will have limits.



You can also right click on individual points to add notes.

You can do the same thing for each sample definition. Just make sure the appropriate settings are in place, samples are collected, and the correct control chart is selected.

Easy to Update

It’s easy to keep information in your SPC control charts up-to-date. You can simply add notes to samples as they are taken, and add assignable causes to samples whenever you need to. Charts are automatically updated with every new sample, so you can always be sure you see the most accurate data, when you need it.

8.4 Recipe Management

New to Recipe & Changeover?

Download and testdrive this most powerful MES solution available anywhere!

Download and Install Module

A Simple Workflow

Step 1. Database Connection

Step 2. Configuring MES Databases

Step 3. Installing the Production Simulator

Step 4. Production Model Configuration

8.4.1

Recipe & Changeover Module in a nutshell

Click on the topic you would like to learn more about ...

- Components
- Scripting
- Objects

Product Data Sheet

To see the Product Data Sheet,

click on Recipe & Changeover Module

Click here to see Knowledge base articles of **Recipe & Changeover**.

Info

Sepasoft Recipe & Changeover module uses system.recipe functions for scripting.

Read this section about licensing...

Licensing and Activation

8.4.2 Recipe Management Overview

What Is A Recipe?

In a manufacturing environment, a recipe (or process recipe) defines the machine settings used by equipment on a production line to process material and produce product or sub-assemblies in a consistent manner. It is an essential part of the production process. OEM equipment tends to have a built-in recipe editor where recipes can be created and selected, whereas simpler equipment may just have setpoints for temperature or speed that can be entered or adjusted. A process engineer will generally dial in the settings during new product introduction and qualify those settings as the Process Of Record (POR) for a given product on a given line. When the production line is setup to run a certain product, these qualified settings need to be setup on the equipment. This can be a tedious task and prone to mis-processing if setpoints are not set correctly. Even if a recipe is selected on a piece of equipment, the settings in that recipe may have been adjusted for a maintenance run or out-of-spec raw material and may no longer be valid for the product.

The management of process recipes attempts to prevent mis-processing by forcing the download of POR recipe setpoints from the Manufacturing Execution System (MES). This provides an efficient means to manage and select recipes, track variances in recipe values, keep recipes secure, track recipe changes and generate recipe reports.

Recipe Module Overview

The Recipe Module manages and monitors recipes, and is ideal for quickly and accurately changing machine settings or process recipes. Powerful master recipe and sub-recipe management, recipe security, change log tracking, variance tracking and more empower you take more control of your manufacturing process. The module provides a rich set of components and functions that reduces the time required to manage production recipes and track process variances. In a nutshell, the modules provides...

- Master Recipe Management
- Sub-Recipe Management
- Recipe Security
- Change Log Tracking
- Variance Tracking
- Recipe Scaling

This module helps enforce precision automation that prevent faults, increases throughput and yield, and adjusts processing in order to prolong the time between required maintenance activities. It is ideal for quickly and accurately changing machine, process or system recipes. Powerful master recipe and sub-recipe management, recipe security, change log tracking, variance tracking and more empower you to improve efficiency and quality, and take more control of your manufacturing facility.

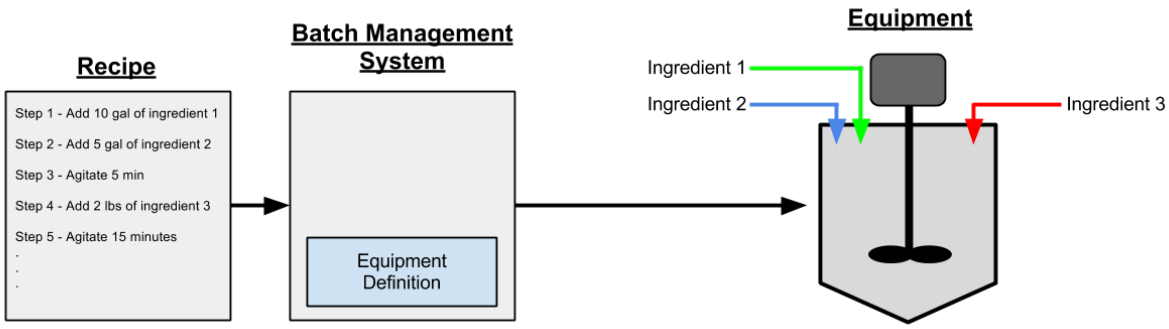
Recipe Types

Batch Recipes

We commonly think of recipes as making a batch of product. An analogy to this is a batch of cookies where many ingredients are added in sequence along with mixing. It is important to understand that a batch system is different from a recipe. It is true that batch systems use recipes, but a batch system has equipment definitions that are combined with the recipe to control the machinery to make a batch of product.

Batch Management Systems handle many other functions including inventory checks before starting a batch, alarm detection, machine control and more.

The Recipe / Changeover Module does not do the functions of a Batch Management System. This being said, you can add multiple steps as child recipes to a master recipe and then step or sequence through the steps. The sequencing through the steps must be done in script or the PLC.



Batch System Diagram

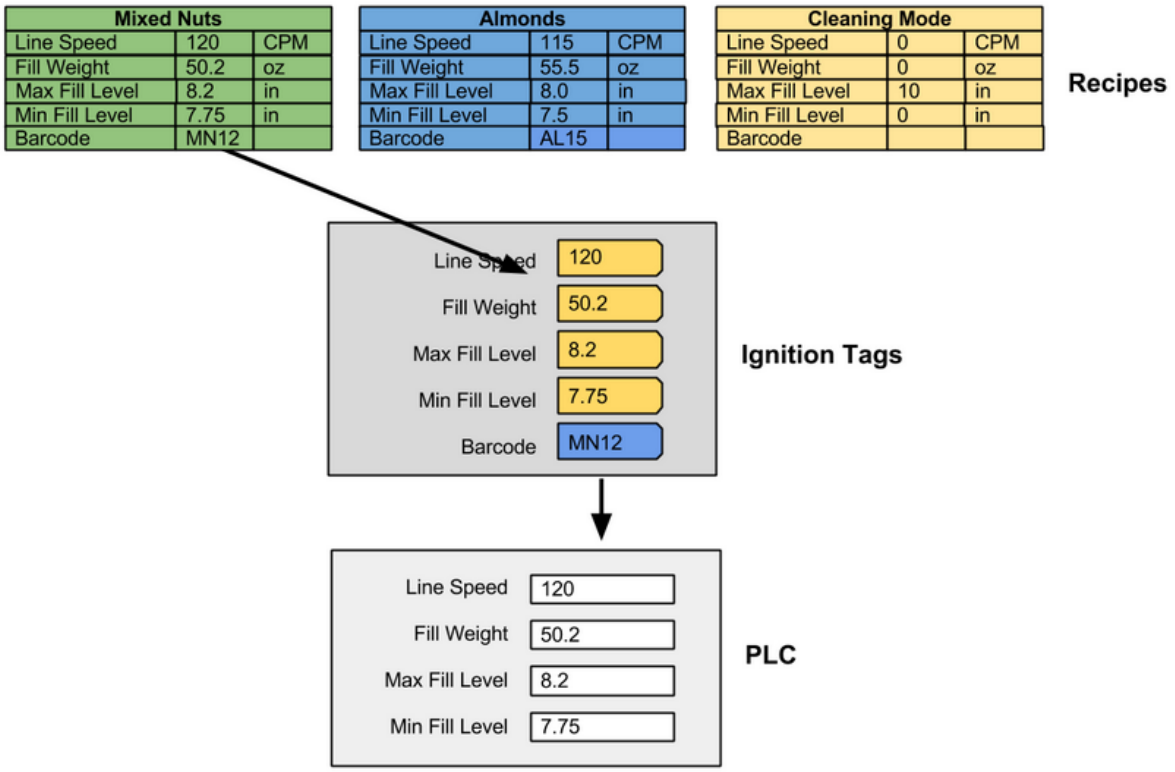
Batch System

Machine Recipes

Machine recipes are used to setup equipment to produce a given product or to place it in a given mode. If a machine can run 20 different products and each product has different settings, then the need to manage recipes is essential. Commonly, machines have some sort of operator interface that will allow the operator to change settings and in some cases, have a very basic recipe system. This can work okay for a single machine but with production lines where there are several machines, it becomes more of a task to go to each machine and make sure it is setup to run the next product on the schedule. This requires time and is prone to mistakes during changeover between products.

When a recipe is selected for a machine, the recipe values are written to Ignition tags which can be mapped to memory locations in a PLC. In the image below, all the recipe values except for the Barcode are mapped to a PLC through OPC. The Barcode recipe value is just mapped to a memory tag and can be displayed on a screen for the operator to verify the barcode number or it can be sent to a printer through serial or TCP/IP.

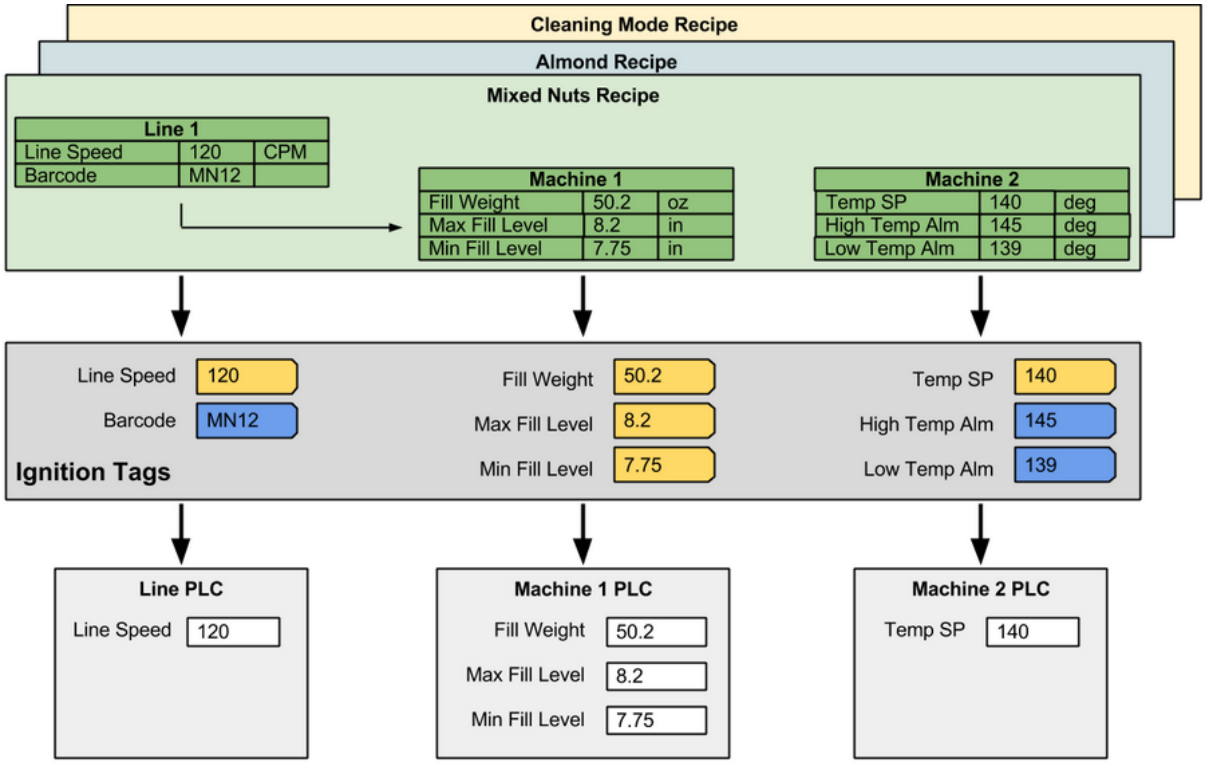
If the Almonds recipe is selected, the recipe value will be written to the tags. If the tags are tied to PLC memory addresses, they will end up in the PLC and the machine will be ready to run almonds.



Single Machine Recipe

Single Machine Recipe Value Tag Mapping

When a production line contains production cells, cell groups or locations as children, the recipes can be managed, selected and reported on by the production line. The image below depicts this where Line 1 has two machines as children. These children can be production cells, cell groups or locations. If the machines are being tracked with the OEE Downtime Module, then the existing production cells or cell groups should be reused. However, if the machines are not being tracked with the OEE Downtime Module, then locations can be used. See Production Model for more information.



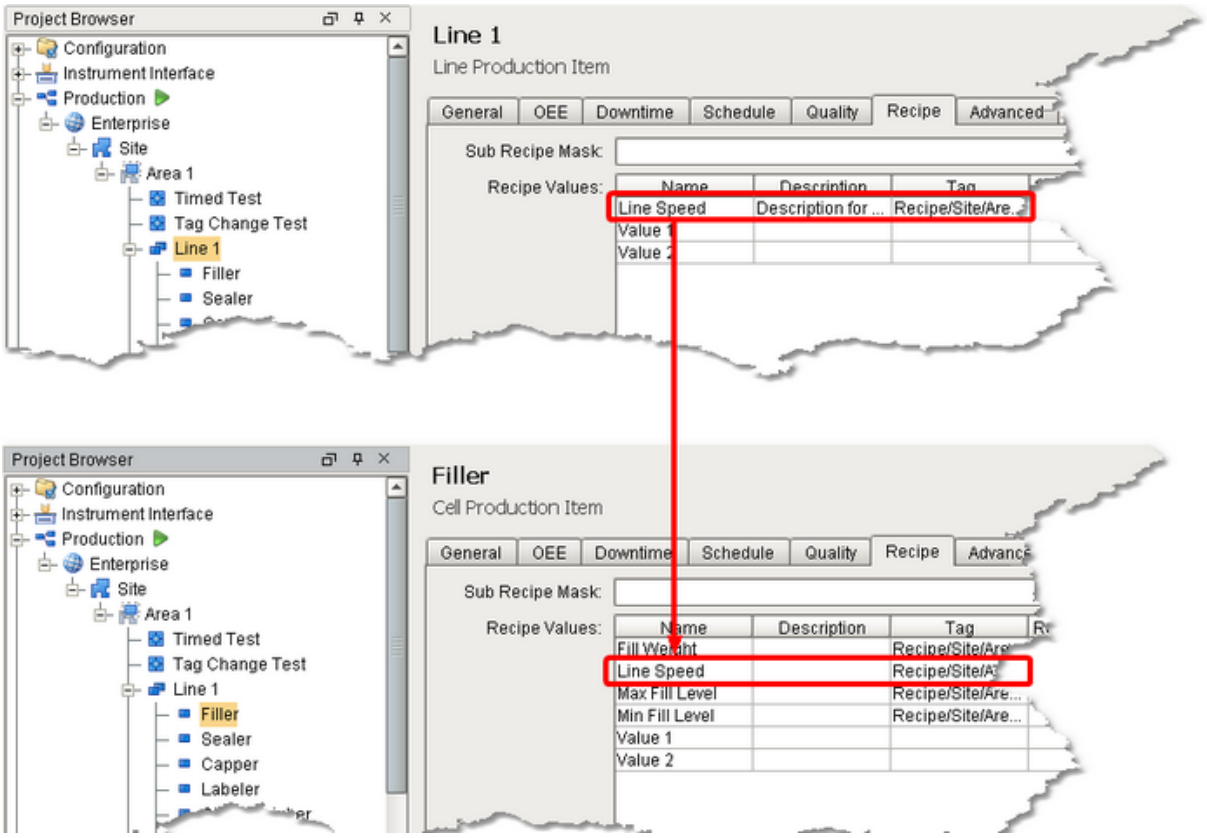
Multiple Machine Recipe Value Tag Mapping

This is a brief overview of the types of recipes that are commonly used in manufacturing. There is a lot more functionality such as scaling, variance monitoring, change logs, master recipes, sub recipes, reporting capabilities, etc. that comes along with the Recipe / Changeover Module that is covered in the following sections of this manual.

Recipe Value Propagation

The recipe management module relies heavily on the Production Model for handling recipe values. **Area**, **Line**, **Cell Group** and **Cell** Production items in the Production model can be configured to inherit recipe values from the parent production item or use their own recipe value. It is important to understand the production model as it is heavily referred to when dealing with recipe values. Recipe values are defined by machine, or in some cases a virtual location. Once recipe values are defined for a machine in the Production model, they can then be added to recipes. See the section below for an overview of the Production Model.

Recipe values that are added to a production item can be propagated down to the child production items, based on the 'Inherit Recipe Values Mode' setting. As an example, if **LineSpeed** recipe value is added to a production line, then all cells, cell groups and locations that are children of the production line, will also have the **LineSpeed** recipe value. The Ignition tag associated with the recipe value is not propagated to the child recipe value.



Recipe Value Propagation 1

Recipe Value Propagation

Only the recipe values that have Ignition tags assigned to them will appear in the recipe editor. So, if a propagated recipe value is not relevant to the child production item, the recipe value Tag property can be left blank.

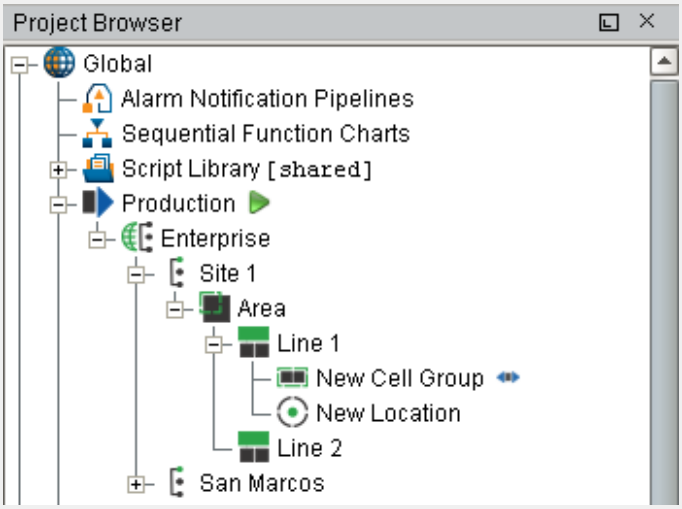
Help Manual: Production Model Configuration

What Is The Production Model?

When any of the core MES modules (OEE, SPC, Recipe, T&T) are installed, the Production Model is added to the **Global** project resources in the Project Browser window of the Ignition Designer. The Production Model allows you to define your manufacturing process in a tree view form and provides an organized way to configure, control, and analyze your manufacturing activities. It provides the foundation on which the MES modules are built.

The Production Model is a hierarchy of Sites, Areas, Lines, Cell Groups, Cells, Locations, Storage Zones and Storage Units. Typically, Lines and Cells are used to represent machinery or equipment where a process occurs transforming raw materials into sub-assemblies or finished goods. Storage Zones and Storage Units are typically used to define where to get or store material.

Lines and Cells defined in the production model should be considered to be equipment that is bolted to the floor and has conduit running to it. Mobile equipment such as pallets, bins, dies used for pressing, etc. are not defined in the production model, but configured in the MES Management screen as Supplemental Equipment (Track & Trace only).



Below are the different types of Production Items that can be added to the production model.

Icon	Production Item	Description	Module
	Enterprise	The enterprise is the highest level of the production model and typically represents a manufacturing company. You can rename the Enterprise production item to your companies name. You can only have one Enterprise item in the Production Model.	All
	Site	A site is a fixed geographical production location that is part of an enterprise. Separating your enterprise into multiple production sites allows for comparing OEE, downtime and production information between them.	All
	Area		All

Icon	Production Item	Description	Module
		An area is a physical or logical grouping of production lines.	
	Line	A line is a collection of one or more cells and/or cell groups that work together to perform a sequence of process steps. Typically, the product flows from one cell or cell group to the next in sequence until the product, or sub assembly, being produced is complete. Understanding how Operations schedules or controls a production run will help in determining whether cells should be grouped into a line or be considered lines themselves.	All
	Location	A location item is the place where a sample is collected. This can be placed under an area or a line.	SPC
	Cell Group	A cell group contains two or more cells. Typically, these cells occur at the same time in the sequence of the line instead of one after another, causing the cell group to act as a single sub process or step within the production.	All
	Cell	The cell is a single machine, sub process or step required in the manufacture of a product. The product may be a hard product such as used in packaging, adding liquid or powder, etc. Packaging machines are a common example, but a cell applies to processes also.	All
	Storage Zone	A storage area such as a warehouse.	T&T
	Storage Unit	A storage unit located inside of a storage zone. For example, you may have a warehouse with bay 1 to 5.	T&T

Configuring the Production Model

The production model is configured within the Ignition designer and is accessed by selecting the **Production** node under Global in the project browser. From here your enterprise, site, area(s), line(s) and line cell(s), line cell group(s), storage zone(s) and storage unit(s) can be added, renamed and deleted.

> It is extremely important to understand production OPC values have an OPC item path that matches the layout of the production model and that renaming production items can cause Ignition tags associated with a production item to stop being updated.

Adding a New Production Item

To add a new Production item, right-click on the Production model and select the **New Production Item** > **New Production xxxx** menu item.

Renaming a Production Item

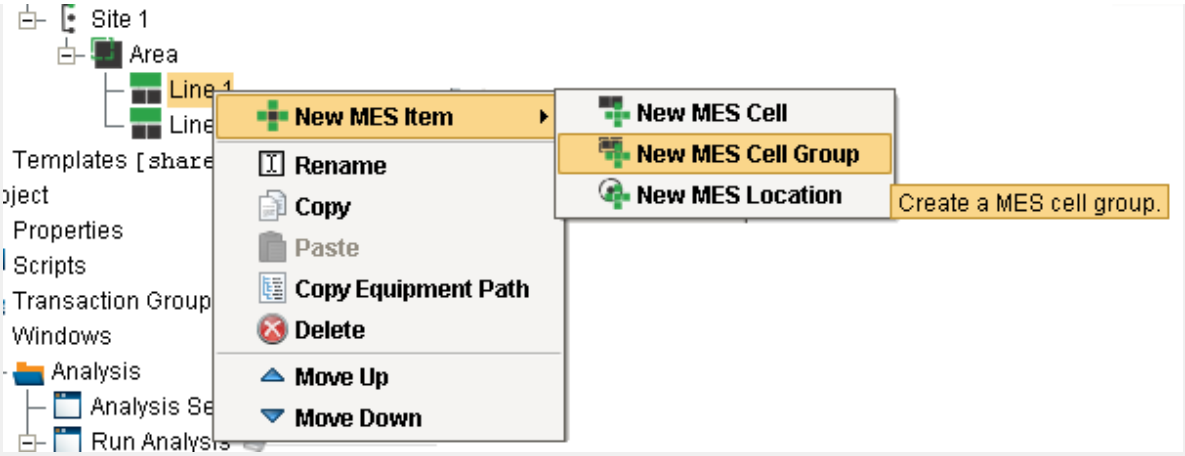
To rename a production item, right-click on it and select **Rename**, then enter the new name.

> Please note that when you rename a production item, it actually creates a new instance of a production item and disables the old production item. This is important to note as data captured against that production item will not be accessible to the newly renamed production item. Spend the time to get the Production Item named correctly at the beginning of the project.

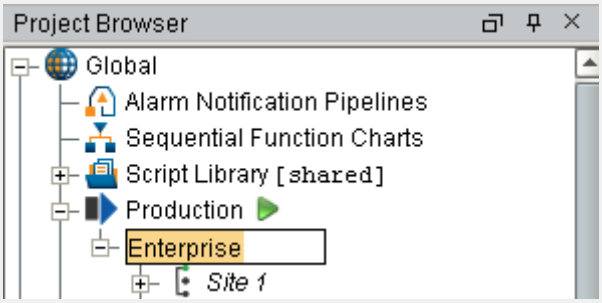
Deleting a Production Item

To remove an existing production item, right-click on the item and select the **Delete** menu item. A window will appear confirming that you permanently want to delete the production item.

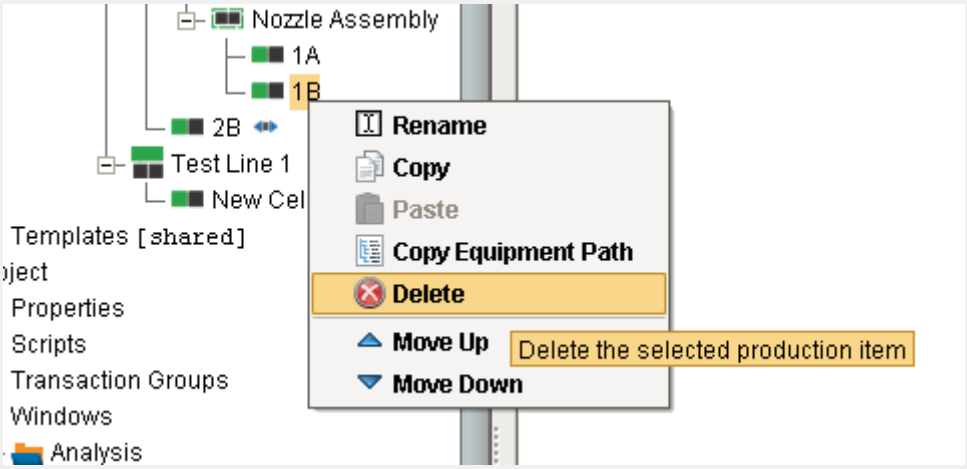
> Please note that any line(s), cell(s), cell group(s) and location(s) underneath the production item will also be permanently removed.



Adding a new Cell Group to the Production Model



Renaming the Enterprise



Delete a Cell

Copying a Production Item

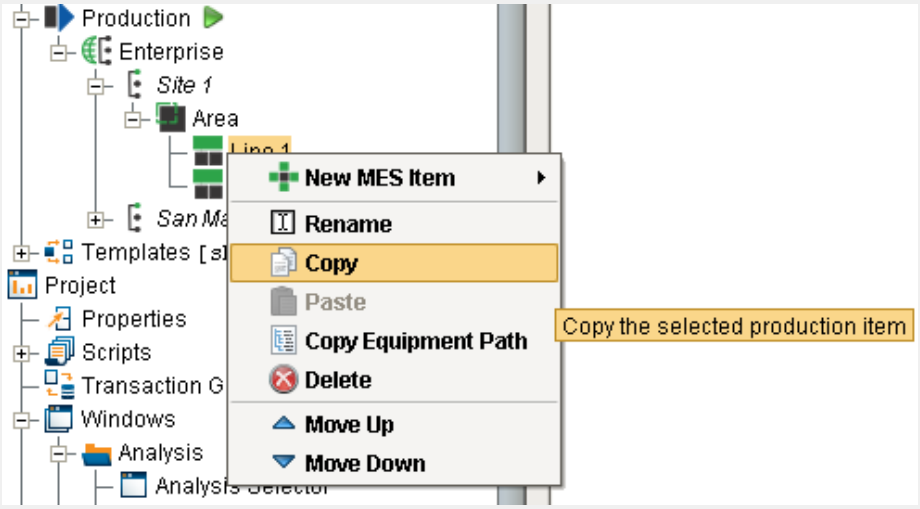
Right Click mouse button and select **Copy** on any production item to copy that production item.

Right Click mouse button and select **Paste** to make a copy of that production item in the production model.

If you are copying a line, select the line before copying it. When you paste it, select the area in which to the create a copy of that line.

Good Practice

It is recommended that you make a gateway backup prior to copying and pasting Production items. It is not recommended that you make changes to the production model on the production server without scheduling with Operations and having the system backed up.



Copying a Production Item

Production Item General Settings

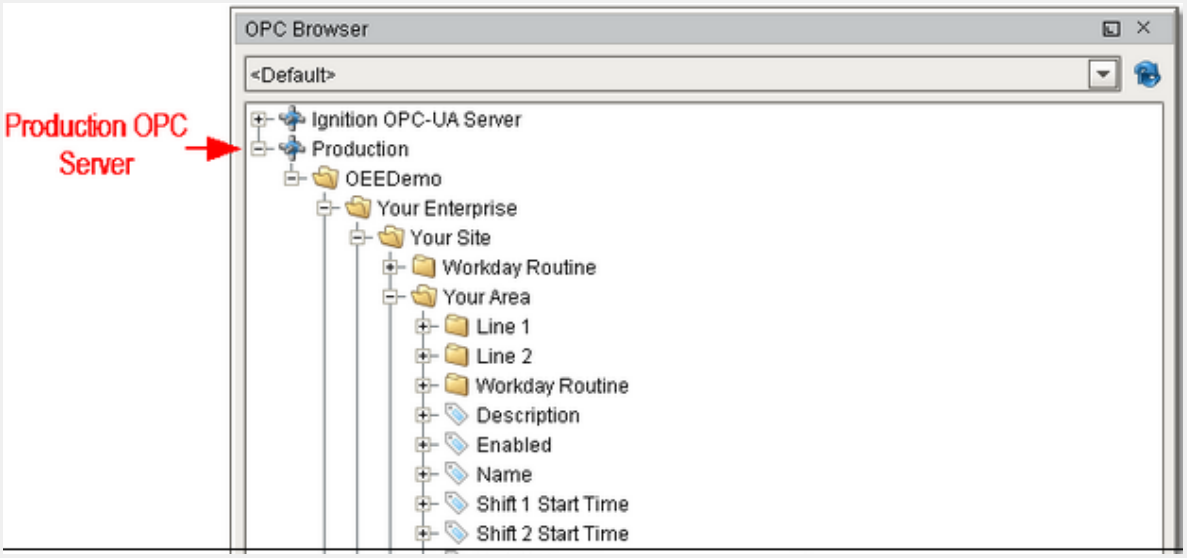
The general settings are accessed by selecting the desired production item and selecting the General tab.

Setting	Description
Enabled	By default, the production item is enabled. It can be disabled by un-checking the Enabled setting and saving the project. This will stop the track and trace, OEE, downtime, SPC, recipe and scheduling modules from using the area and any other production items that are underneath it.
Description	This is an optional description and is just for your reference.

OPC Production Tags

As production items are added to the production model, run time access into configuration settings and current state of those production items is available through the Production OPC Server. It is added automatically when MES Modules are installed. When the production items are added, removed or modified, the changes will be reflected in the Production OPC Server when the project is saved in the designer.

Please refer to the OPC Production Server Tag Reference in the Appendix for more help.

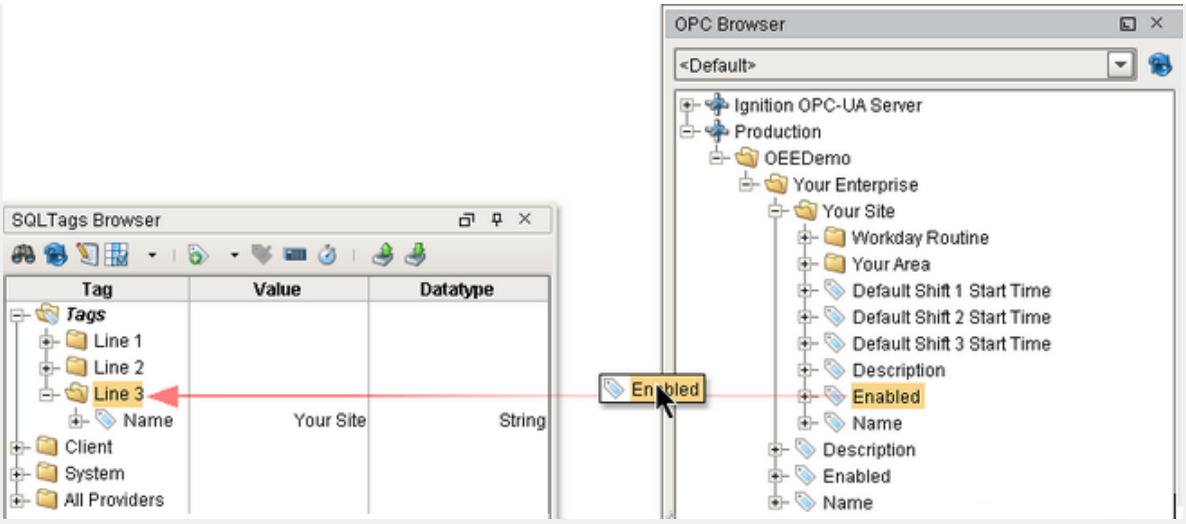


Demo OPC Values

Using OPC Production Tags in Your Project

Before Production OPC Server tags can be used in your project windows, transaction groups etc., they must be added to the Ignition SQLTags. This is done in the designer by selecting the SQLTag Browser and clicking on the OPC icon. This will cause the OPC Browser to appear. Next, drill down in the Production node within the OPC Browser. Drag the desired Production OPC Values over to the SQLTag Browser as shown.

When writing to OPC values that are related to production model settings, the new value is not retained upon restarting. This is because production model settings are saved in the Ignition project and is only saved when done so in the designer.



Add Production OPC Server Values to SQLTags

Default Values

When a new recipe is created, it is initialized with a default value. If the recipe value is assigned to an Ignition tag that is tied to a PLC memory address, then the default value should be what is normal and default for the machine. However, it can be any value you want as long as it is within the range of the data type for the tag the recipe value is associated with and is within the security settings for the recipe value. See Recipe Security for more information.

Inside a recipe, a recipe value can use the default value or it can be overridden in the recipe as shown in the image below. Notice the Agitator Speed and Ingredient 2 of the Thick Blend recipe have been overridden. If it uses the default value, it will be updated when the default value is changed. Once a recipe value has been overridden in a recipe, that value persists even if the master value is changed. At any time the recipe value can be reverted back to the default value.

Default Values

Mixer	
Value Name	Default Value
Agitator Speed	35
Mix Time	20
Ingredient 1	15
Ingredient 2	50.2
Ingredient 3	8.2

Recipes

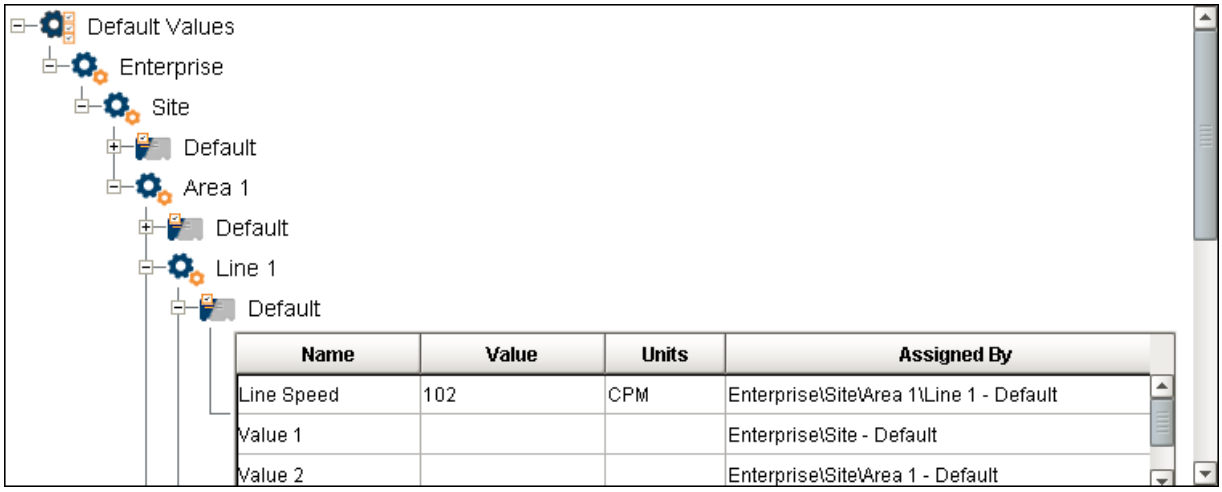
Creamy Blend		
Value Name	Recipe Value	Assigned By
Agitator Speed	35	Mixer-Default
Mix Time	20	Mixer-Default
Ingredient 1	15	Mixer-Default
Ingredient 2	50.2	Mixer-Default
Ingredient 3	8.2	Mixer-Default

Thick Blend		
Value Name	Recipe Value	Assigned By
Agitator Speed	37	Thick Blend
Mix Time	20	Mixer-Default
Ingredient 1	15	Mixer-Default
Ingredient 2	55.0	Thick Blend
Ingredient 3	8.2	Mixer-Default

Recipe Default Value

Recipe Default Values

All of this is done in the recipe editor or by using script functions. The image below shows the default values in the recipe editor where the default values can be edited. Also see Sub Recipes for more details of how default values are used with sub recipes. The section on Recipe Security provides more detail on changing the security for recipe values that are accessed in the default values.



Default Values in the Recipe Editor

Master Recipes

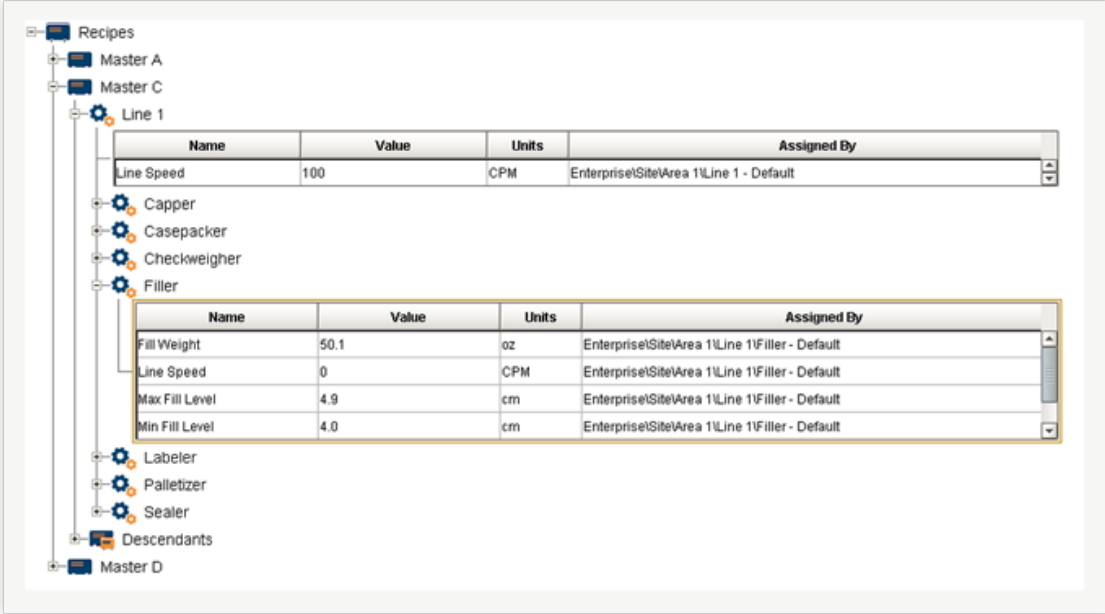
Making a change to a recipe value that is used in numerous recipes is a daunting task and is prone to mistakes. To address this problem the Recipe / Changeover Module uses master recipes. The image below shows two recipes that are derived from, or descendents of, the Master Blend recipe. When the descendant recipe is added, all recipe values will be inherited from the master recipe. When a value is changed in a descendant recipe, it will override the value from the master recipe with the new value as shown in the image below for the Agitator Speed and Ingredient 2 recipe values.

The Sepasoft Recipe Management Module reduces the effort required to manage numerous recipes with master recipe functionality. When you change a setting in the master recipe, it will replicate down to all of its sub-recipes while still maintaining the specific values of each sub-recipe. With unlimited levels available for master recipes, you can organize recipes in a hierarchical manner, greatly reducing the effort to maintain recipes.

Simple Recipe Editing

Managing recipes has never been easier using the Sepasoft Recipe / Changeover Module's built-in visual recipe editor. The following editing capabilities are now just a mouse-click away:

- Create new recipes
- Read current values into a recipe
- Export recipes
- Import recipes
- Manage security
- Select machines for recipes
- And more



Complete Recipe Scripting

The Sepasoft Recipe / Changeover Module comes with complete scripting functions for you to:

- Add recipes
- Change values
- Change a machine's recipe

If you need additional functionality beyond the out-of-box functionality of the Recipe / Changeover Module, you can use the script functions to accommodate your production environment, instead of changing your production environment because of software limitations.

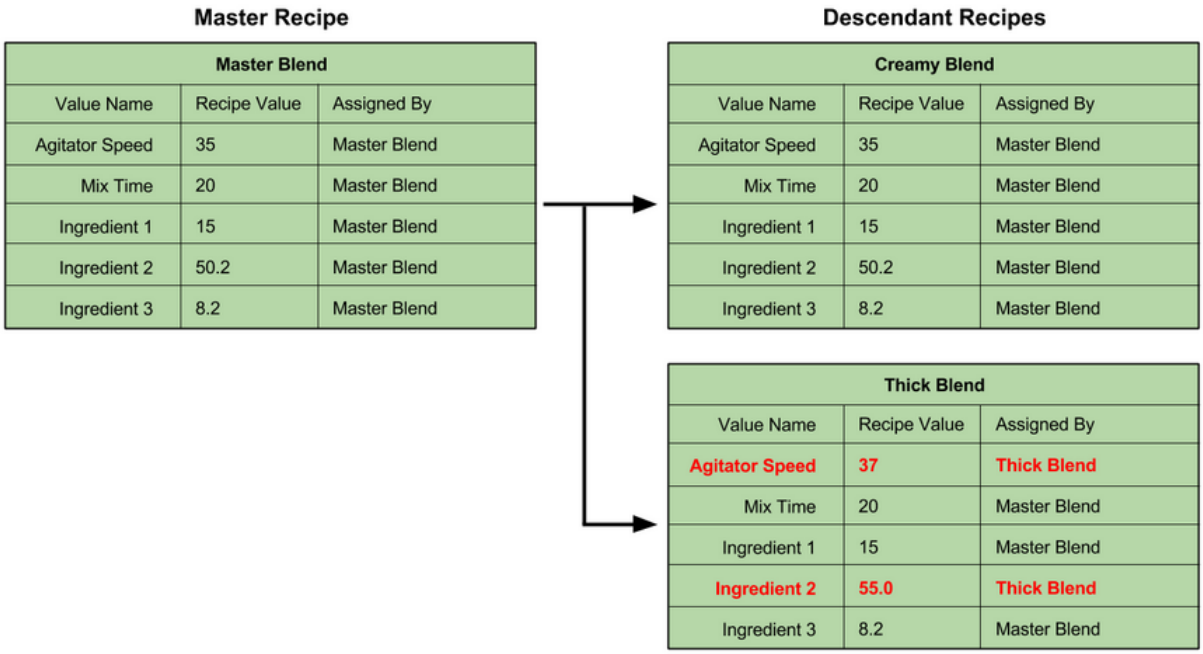
OEE and SPC Integration

In a production process that fully employs the Sepasoft MES suite of modules, making a single product code selection sets recipe values, starts OEE (overall equipment effectiveness) tracking and collects SPC (statistical process control) samples. During and after the production run, the Sepasoft Recipe / Changeover Module enables you to analyze all of the following data in one unified system:

- Recipe data
- Production data
- SPC data
- And more

The image below just shows one master recipe and two descendant recipes. In actual fact, there can be any number of levels of master recipes and any number of descendants of a master recipe. Any recipe that has descendants is considered a master recipe. Consider a master recipe called Master 1 that has a descendant that is called Master 1-A that has a descendant called Final 1-A-A. Then recipes Master 1 and Master 1-A are both master recipes and recipe Final 1-A-A is a final recipe. Only final recipes can be selected for a production line, cell, cell group or location. See Selecting Recipes for more information.

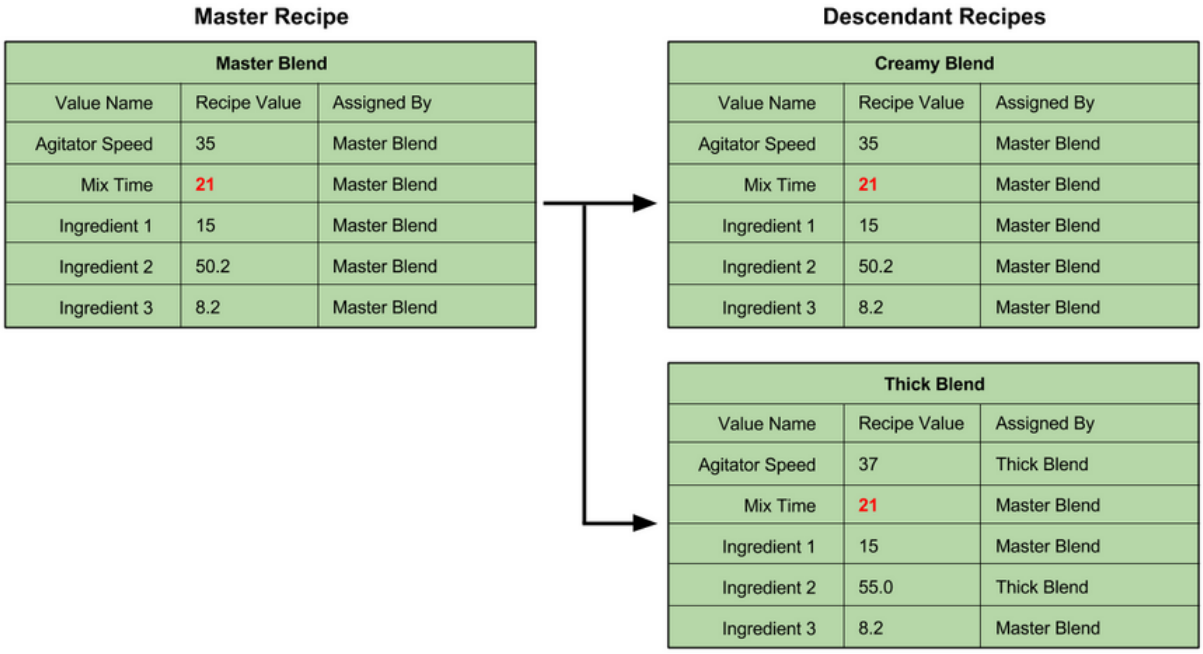
One aspect that is not shown in the image below is that the master recipe can inherit its values from the default values of the associated production item. So the production item has its default values, which is added to a recipe so the recipe inherits from the default values, then the descendant recipes inherit from the master recipe and so on. That is until a recipe value is overridden somewhere along the inheritance chain. See Default Values for more information.



Recipe Master

Master Recipe

When a value is changed in the master recipe, it is propagated down to the descendant recipes. As shown in the image below, the Mix Time recipe value is changed to 21 and the Creamy Blend and Thick Blend recipes also reflect the new value.



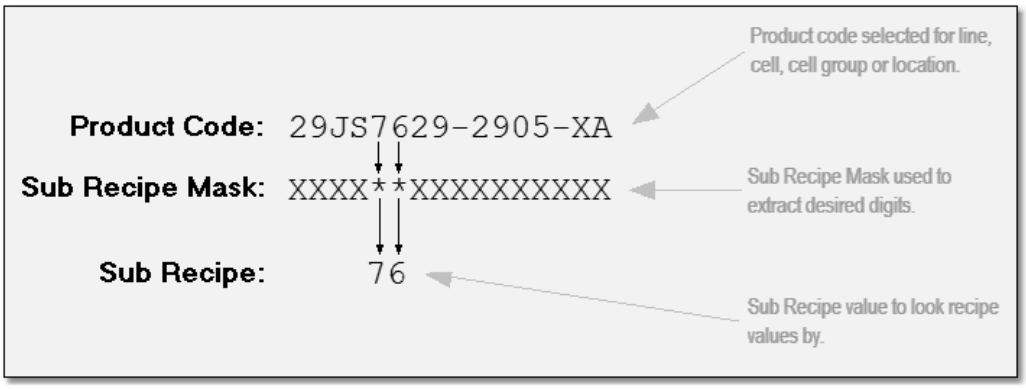
Recipe Master Change

Master Recipe Value Change

Sub Recipes

Sub Recipes are convenient when a machine's recipe can be determined from digits within a product code. When the product code is used as the recipe, a portion of the product code can be extracted and used to determine the machine's recipe. For example, you may have a tape machine that recipe values only change based on the case size. If there are only two different case sizes and there is a digit in the product code that specifies the case size, then sub recipes can be used for the tape machine. All other machines can use the normal recipe functionality.

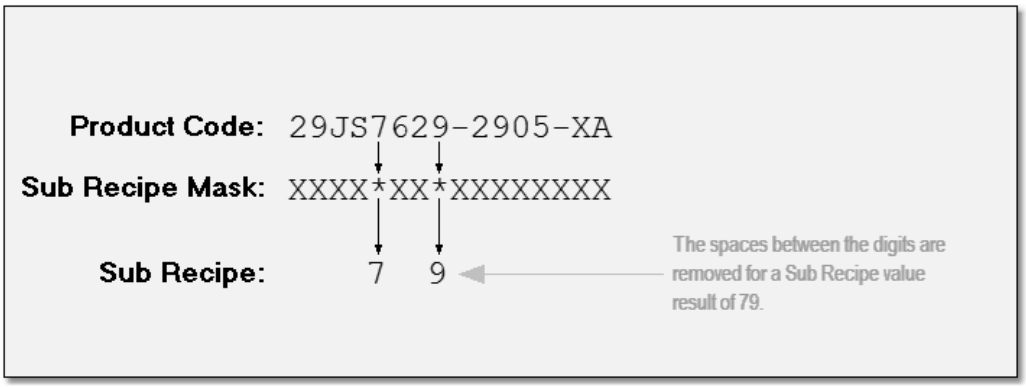
Sub recipes are derived from the product code and the sub recipe mask. The sub recipe mask specifies the digits to extract from the product code to determine the sub recipe. Once the sub recipe value is determined like the 76 in the image below, the recipe values are looked up in the sub recipes for the production line, cell, cell group or location and are written to tags. See Sub Recipe Mask for more information on how to configure production items to use sub recipes.



SubRecipeMask

Sub Recipe Mask

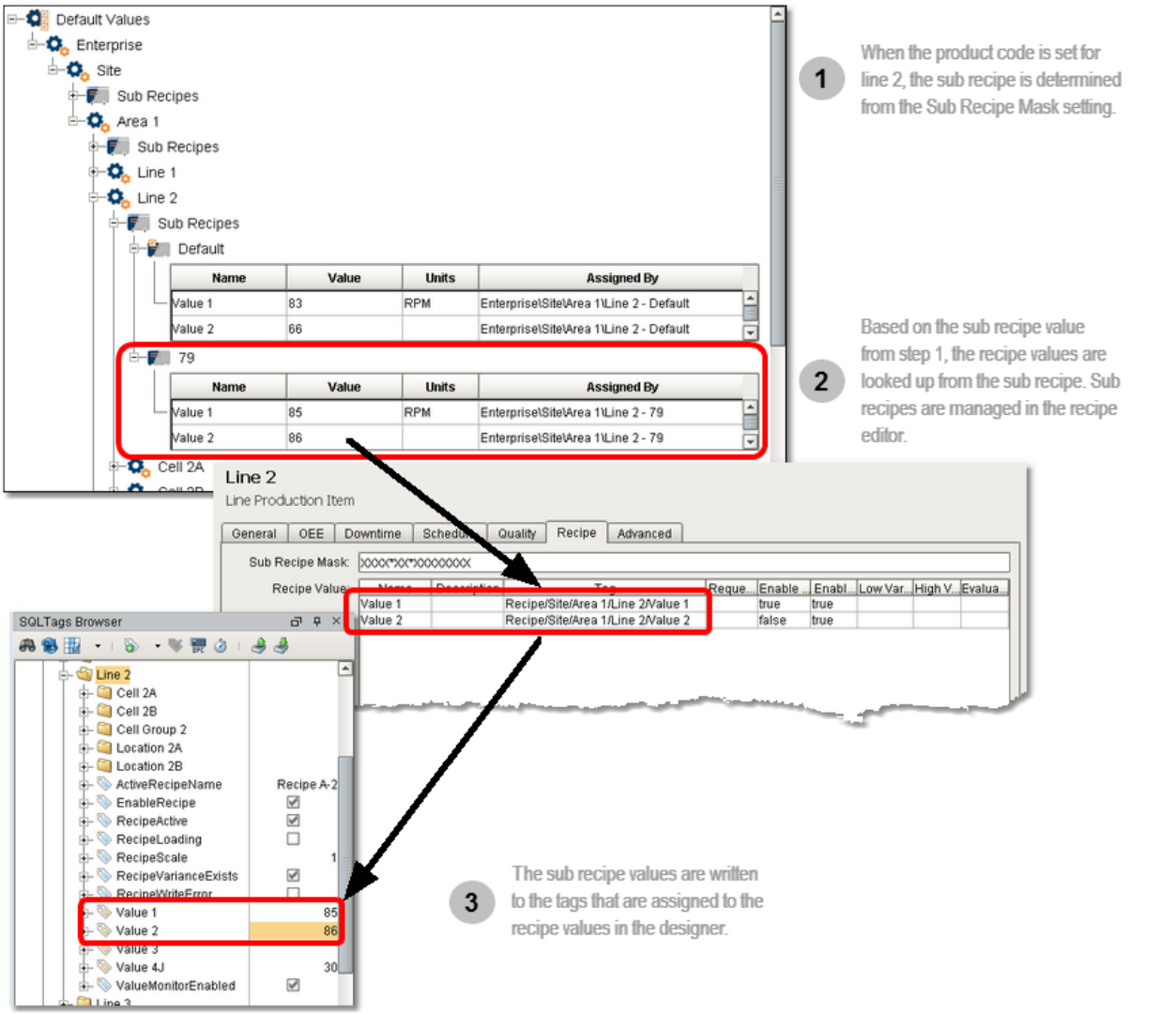
There are two different mask characters that should be used in the Sub Recipe Mask. The first is just a placeholder and any characters that exist in the corresponding digit position of the product code will be ignored. The other mask character is the asterisk and any characters that exist in the corresponding digit position of the product code will be used in the sub recipe value. The asterisk characters in the Sub Recipe Mask do not need to be in consecutive digit positions as shown in the image below.



Sub Recipe Mask with Non-Consecutive Asterisk Mask Characters

The image below steps through the flow of selecting sub recipes and setting the associated tag values and is based on the determining of the sub recipe as described above. The product code can be selected using various methods. It can be selected using the Recipe Selector List component, but is can also be selected by starting a production run for the OEE Module or by using one of the script functions. In fact, it can be selected using a combination of methods.

The Recipe Editor component is used to edit both normal recipes and sub recipes. See sub recipes in the Editing Recipes for more details. In general, the recipe editor is used to manage sub recipes for a line, cell, cell group or location. New sub recipes can be created and the recipe values for each can be edited.



SubRecipeMask3

If a sub recipe is not found, the default will be used.

Recipe Change Log

Keeping an audit log of when recipes are changed, by who and why can be important. Especially in some industries where regulatory compliance are in force. The Recipe / Changeover Module records all changes to recipes whether the changes were made from the recipe editor, importing or script. The only changes not automatically detected are changes made directly to the database and proper database security should be implemented if this is a risk.

Below are the methods that the change log history can be examined.

Recipe Change Log Viewer

There is a component that will easily show recipe change log history on screens. It has properties to narrow in on what production item and recipes to show the change log for. The columns that are shown are configurable through the table customizer. See Recipe Change Log Viewer component for more information. The image below depicts the change log viewer.

ValueName	Change...	TimeStamp	ChangeType	FromValue	ToValue
Line Speed	admin	Jun 19, 2013 3:18 PM	Recipe value changed		85
Line Speed	admin	Jun 19, 2013 3:17 PM	Recipe value reverted	99	
Line Speed	admin	Jun 17, 2013 11:21 AM	Recipe value changed		99

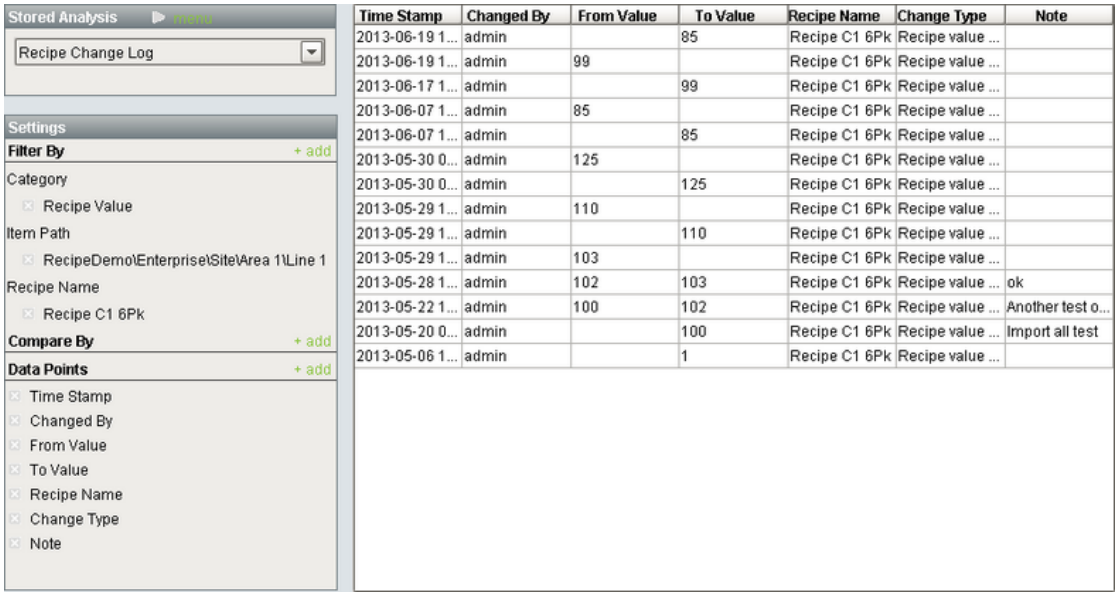
Recipe Change Log Viewer

Recipe Change Log Viewer

Recipe Change Log Analysis Provider

The core production module common to all MES modules has an analysis engine. Each MES module provides an analysis provider to work with the data collected for the specific module. In the case of the Recipe / Changeover Module, there are 4 different analysis providers of which one of them is the Recipe Change Log provider. It is used to query change log history information based on your selections. See Recipe Analysis Provider for more information.

The image below shows the interactive Recipe Change Log Analysis Provider using the core analysis components, but it can also be used to provide data to the Ignition Reporting Module.



Time Stamp	Changed By	From Value	To Value	Recipe Name	Change Type	Note
2013-06-19 1...	admin		85	Recipe C1 6Pk	Recipe value ...	
2013-06-19 1...	admin	99		Recipe C1 6Pk	Recipe value ...	
2013-06-17 1...	admin		99	Recipe C1 6Pk	Recipe value ...	
2013-06-07 1...	admin	85		Recipe C1 6Pk	Recipe value ...	
2013-06-07 1...	admin		85	Recipe C1 6Pk	Recipe value ...	
2013-05-30 0...	admin	125		Recipe C1 6Pk	Recipe value ...	
2013-05-30 0...	admin		125	Recipe C1 6Pk	Recipe value ...	
2013-05-29 1...	admin	110		Recipe C1 6Pk	Recipe value ...	
2013-05-29 1...	admin		110	Recipe C1 6Pk	Recipe value ...	
2013-05-29 1...	admin	103		Recipe C1 6Pk	Recipe value ...	
2013-05-28 1...	admin	102	103	Recipe C1 6Pk	Recipe value ...	ok
2013-05-22 1...	admin	100	102	Recipe C1 6Pk	Recipe value ...	Another test o...
2013-05-20 0...	admin		100	Recipe C1 6Pk	Recipe value ...	Import all test
2013-05-06 1...	admin		1	Recipe C1 6Pk	Recipe value ...	

Recipe Change Log Analysis

Script

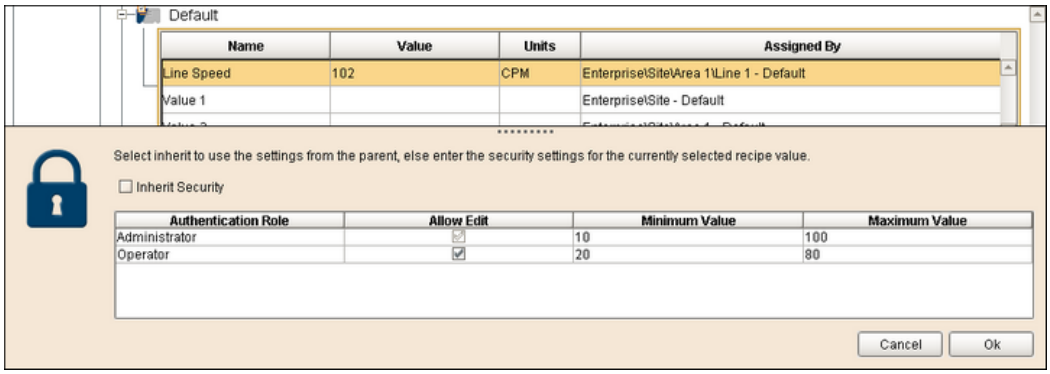
Script functions can also be used to read the recipe change log history. This is useful if the recipe change log history is needed for reasons other than displaying or reporting. The recipe change log history is returned as a dataset from the script and the rows can be iterated through.

See system.recipe.getChangelogHistory script function for more information.

Recipe Security

The recipe value security uses Ignition's authentication roles to limit who can change what recipe values by how much. Each recipe value can be set to specific security settings or it can inherit from its parent. Like other recipes value settings, the security settings can propagate down multiple levels of inheritance.

Referring to the image below, the Inherit Security check box determines if the recipe value should use its parent's security settings or break the inheritance. By unselecting the Inherit Security check box, the settings for each authentication role can be made. Initially when doing so, the inherited security settings will remain that of the parent until they are edited.



Recipe Value Security

Recipe Value Security Settings

The recipe value security is verified when changing values using the recipe editor component, importing recipes or changing values using script.

When changing a recipe value using the recipe editor component, importing recipe values or from client script, the authentication role applied comes from the roles the currently logged in user belongs to. If the user belongs to multiple roles then the role with the least security will be applied. For example, if a user belongs to both the Operator and Maintenance authentication roles, then the least secure one will be applied. If the Operator role can change the Product Pressure recipe value from 10 to 15 and the Maintenance role can change it from 5 to 20, then the Maintenance role will apply.

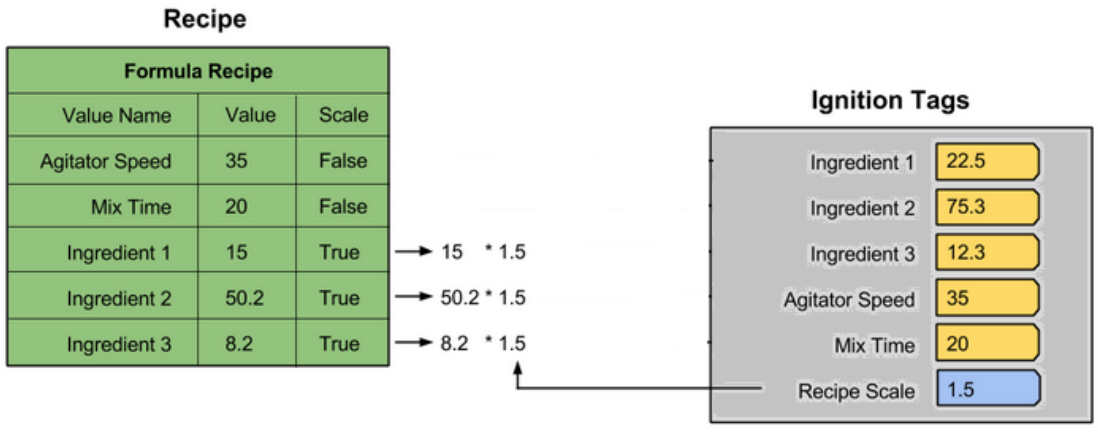
When changing a recipe value from gateway script, the Administrator authentication role is always applied.

The only place the recipe value security can be changed is by using the recipe editor component. Also, it can only be changed in the default values area and not in the actual recipes. Whether or not the logged in user can change the security settings can be controlled with the Enable Security Editing property of the recipe editor component. This property can be bound to an expression to determine if the currently logged in user belongs to authentication roles that are allow to edit security. Another approach is to create a window that allows the recipe value security editing and restrict opening the windows based on authentication roles the currently logged in user belongs to.

Recipe Scaling

When using recipes for batch or other processes that can change based on the amount that is produced, recipe scaling will adjust recipe values based on a recipe scale value. In the recipe value configuration, there is an Enable Scaling setting that can be selected. If the Enable Scaling setting is selected for a recipe value, then whenever a recipe is selected for a production line, cell, cell group or location, the value from the recipe will be scaled by the value in the RecipeScale tag as shown in the image below.

Enabling recipe scaling is done for each individual recipe value. This supports scaling some of the recipe values while not scaling others as might be the case in the example shown below. By default, each production item's Enable Scaling setting is false and must be selected before the RecipeScale value will be applied.



Recipe Scaling

The RecipeScale is a production OPC item that exists for each production line, cell, cell group or location. By default, the RecipeScale is 1.0 and recipe values will not change when recipes are selected. When selecting a recipe for a line, all of the cells, cell groups and locations beneath the line will also be set to the same recipe provided they are enabled. Also, each cell,

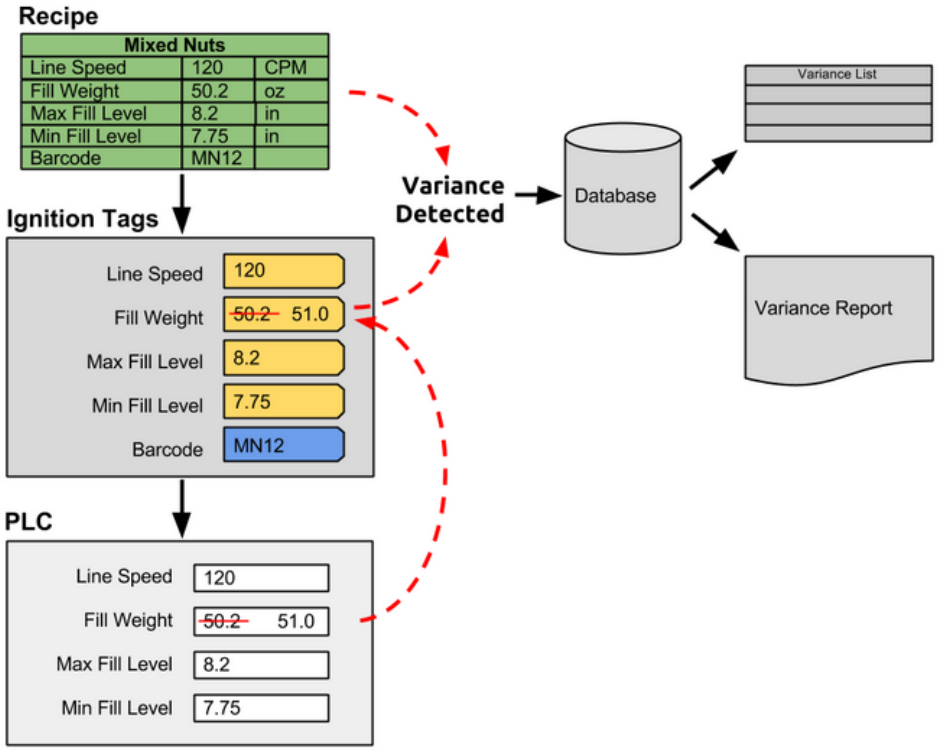
cell group and location RecipeScale value will be set to match that of the line. This enables simple recipe selection for a line without the tedious task of selecting each machine underneath the line.

Variance Monitoring

In most manufacturing systems it is important to know if the live production values match the recipe values. There are two cases where this is important. The first is when the recipe values are first written to verify that they match. The second is during production in the event the live production values changed from an outside source.

Recipe values are written once when the recipe is first selected. It is very important to confirm that the values were successfully set. In the case of the Recipe / Changeover Module, when a recipe is selected, the values are written to the Ignition tags. This should happen successfully, but there can be expressions, scripts, etc. that prevent the value from being written correctly. This is more of an issue when the Ignition tag is configured as an OPC item connecting it to the PLC or other device. If a communication error occurs when the new recipe value was being written to the PLC or device, then it is very useful to know this before machinery is started.

It is very common to have operator interface terminals (OIT) or a standalone human machine interface (HMI) local to a machine that settings can be changed locally. Settings can also be changed from other sources besides the local OIT, and it is important to detect and log when any setting varies from the recipe value.



Recipe Variance Monitoring

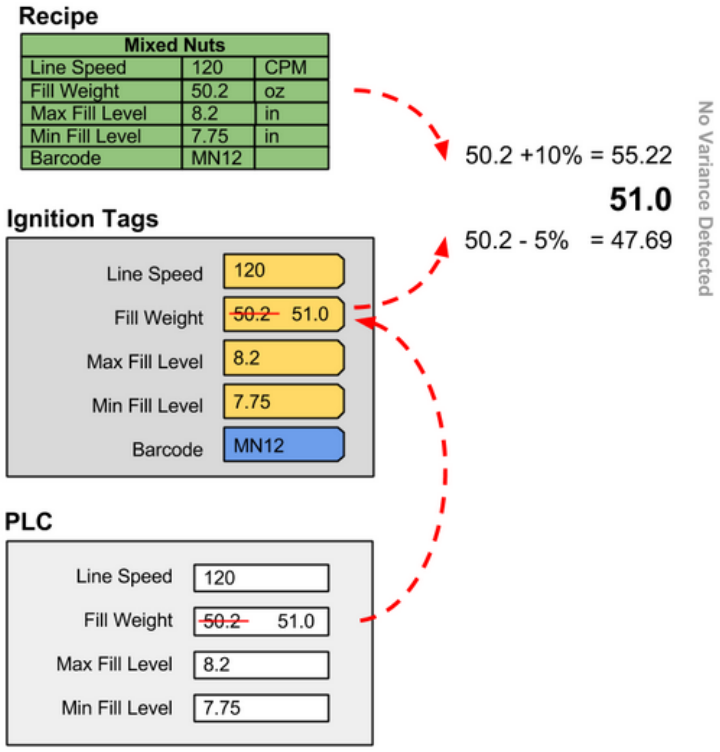
Recipe Variance Detection

Recipe Value Variance Options

There are cases where it is normal for a live production value to vary after the initial recipe value has been written to the Ignition tag. In other cases, it might be okay for the live production value to change within a range. Recipe values in the Recipe / Changeover Module can be configured to not monitor variances or to have a variance window that the live production value must fall outside of before the variance is logged.

By default the variance monitoring is enabled for each recipe value but it can be disabled by recipe value in the designer. This allows for a mix of recipe values that variances will be monitored and other that will not to prevent irrelevant variances from being logged.

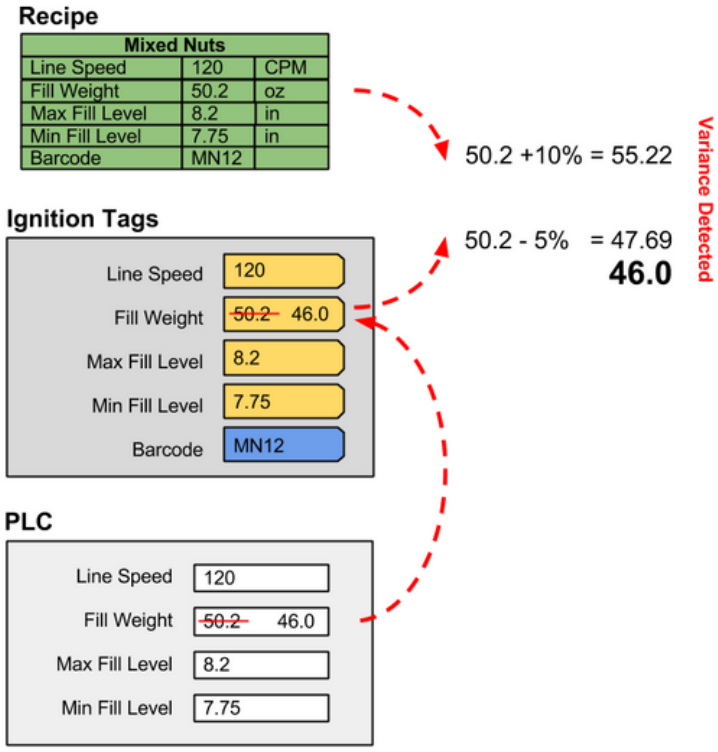
The configuration for the variance window is also done by recipe value in the designer. Both the upper and lower variance thresholds can be defined by percentage of the recipe value or a fixed offset around the recipe value or just fixed values. The image below is showing the Fill Weight recipe value with an upper variance threshold of +10%. The upper threshold value is calculated by starting with the recipe value of 50.2 and adding 10%. The lower threshold is calculated in the same manner. When the Ignition tag value changes, a check is done to see if the current value is between the upper and lower threshold values and in this case as shown in the image below, we see that the current value of 51.0 is between 55.22 and 47.69. As a result, no variance will be logged.



Recipe Variance Range 1

Recipe Value Inside Range

Now lets take a look at a case where the current value is 46.0 as shown in the image below. The value 46.0 is less than the lower threshold and a variance will be logged.



Recipe Variance Range 2

Recipe Value Outside Range

The example above shows the upper and lower threshold values being calculated as a percentage of the recipe value, but they can also be a fixed offset around the recipe value. To configure a recipe value for a fixed offset around the recipe value, an upper variance threshold setting of +<offset> is used. An example is a variance threshold offset of +7.5 were the upper threshold is calculated by adding 7.5 to the recipe value. Using the recipe value from above of 50.2 and adding 7.5 to it will give us an upper threshold of 57.7. The lower variance threshold works the same way.

Instead of the thresholds being calculated as a percentage or fixed offset around the recipe value, fixed values can also be used. For example, a recipe value can be configured with an upper variance threshold of 52.0. In this case, the upper threshold will always be 52.0 irregardless of the recipe value.

Lastly, the upper and / or lower threshold can be calculated using Python script. This is configured in the designer and a recipe value variance range can refer to other tag values, values from databases and much more when calculating the upper or lower threshold values.

Example Evaluate Variance Script

```
upperValue = system.tag.read("[Default]SomeOtherTag")
recipeValue = event.getRecipeTag().getCurrentValue()
if recipeValue > upperValue.value:
```

```
  event.setLogVariance(True)
else:
  event.setLogVariance(False)
```

The script is passed on Evaluate Variance Script object that allows accessing the current tag information and also allows setting the log variance flag. In the script above, a tag called SomeOtherTag is read and compared to the current value of the tag associated with the recipe value where this script was defined. If the current value is greater than the value of the SomeOtherTag, then the setLogVariance method is called with True meaning the recipe value is in a variance state. Otherwise, false is returned.

See Enable Variance Monitoring section for details about configuring recipe values.

Variance Status

As mentioned above, variances are logged to the database and can be viewed in the Recipe Variance Viewer component, analysis and reports. But, having a tag that indicates if any recipe values are in variance for a machine is useful. The Sepasoft MES Modules exposes current status and allows some control through the Production OPC Server. One of the status values provided by the Recipe / Changeover Module is the RecipeVarianceExists value. Each production line, cell, cell group and location has an associated RecipeVarianceExists value. If the value is false, then all live production values are within variance for the production item. If the value is true, then at least one recipe value of the production item is outside of its variance range. See Production OPC Server and Production OPC Values sections for available values that the Recipe / Changeover Module provides.

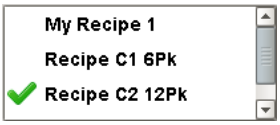
Selecting Recipes

There are different methods of selecting a recipe for production items. Production items that support recipe selection are lines, cells, cell groups and locations. When a production line recipe is selected, it will also select the same recipe for all of the cells, cell groups or locations that are children of the line. This can be disabled by setting the EnableRecipe tag for the child production item to false. This is a feature that makes day-to-day selection of line recipes easier and mistake free.

Recipes can also be canceled that simply turns off variance tracking. This will keep recipe value variance reporting clean with only data from actual production runs.

Components

After the Recipe / Changeover Module is installed, a Recipe tab will be added to component pallete in the Ignition designer. There are two components that allow selection of recipes for a production item. Below is what the Recipe Selection List component looks like.



Recipe Selection List

Recipe Selection List Component

Only final recipes for the production item that is specified by the Item Path property will be displayed. See Master Recipes for more information about final recipes versus master recipes. The list can be limited to only show a subset of recipes by using the Recipe Name Filter property.

End users can select and cancel recipes using this component by right clicking on a recipe and selecting the desired menu item. If this functionality is not desired, then the Read Only property can be set to True to prevent the select popup menu from showing. See Recipe Selector List for more information about the component.

Starting a Trace

Production locations have the ability to trace product that is being process at them. When a production location product code is set using the setLocationProductCode script function, the recipe is also selected for the production location. When this happens the recipe values are written to their associated tags. The location trace can be turn off using the cancelLocationProductCode function.

Currently, scripting is the only method to select a production code for a location, but this will be enhanced when the traceability module is released.

Example

system.production.utils.setLocationProductCode("RecipeDemo\Enterprise\Site\Area 1\Filler Location", "Recipe C1 6Pk", "")

Scripts

Scripts can also be used to directly select and cancel recipes for production items. These scripts supports selecting and cancelling recipes from external triggers such as when a button is clicked or when a product code changes.

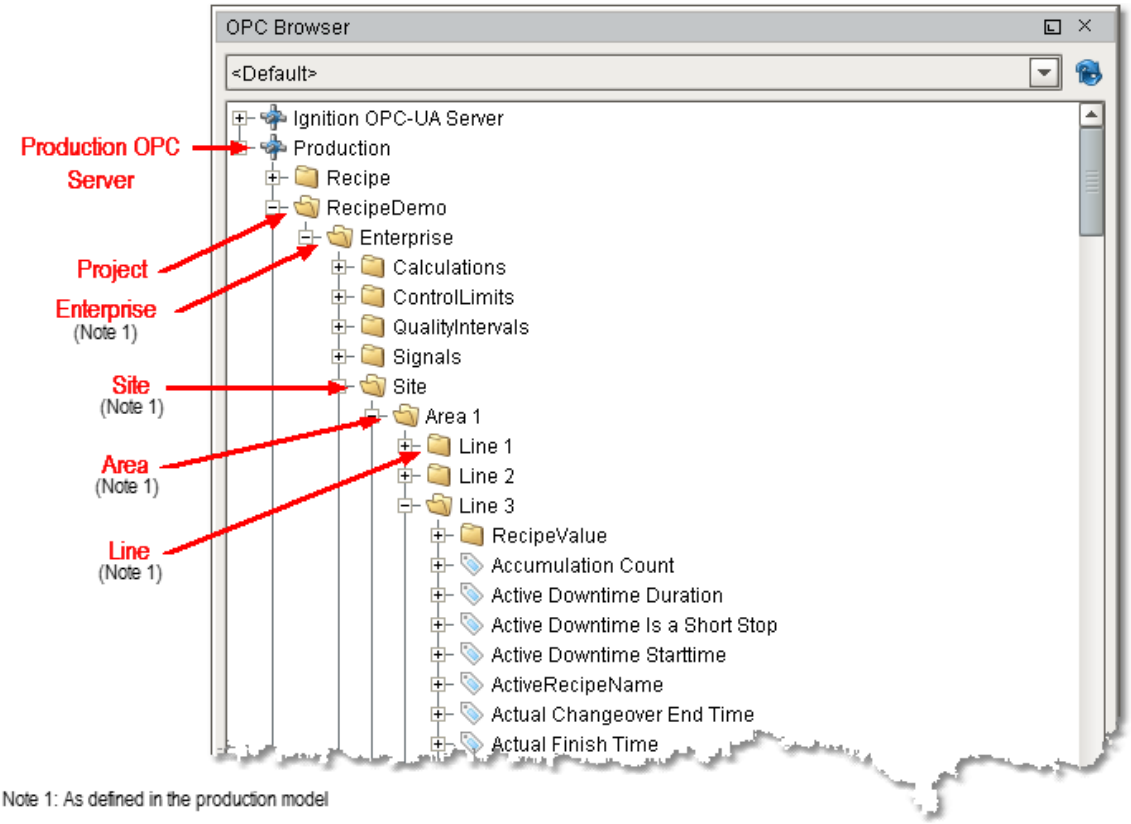
See system.recipe.setItemRecipe script function to select a recipe for a production item and system.recipe.cancelItemRecipe script function to cancel the current recipe for a production item for more details.

Production OPC Server

The production model is defined in the Ignition designer and contains your production lines, cells, cell groups and locations. Runtime access into configuration and current state of the production model is available through the Production OPC Server. It is added automatically

when any of the Sepasoft MES modules are installed. When the production items are added, removed or modified, the changes will be reflected in the Production OPC Server when the project is saved and published in the designer.

Below are some of the values available to read, and in some cases write to for the RecipeDemo project.



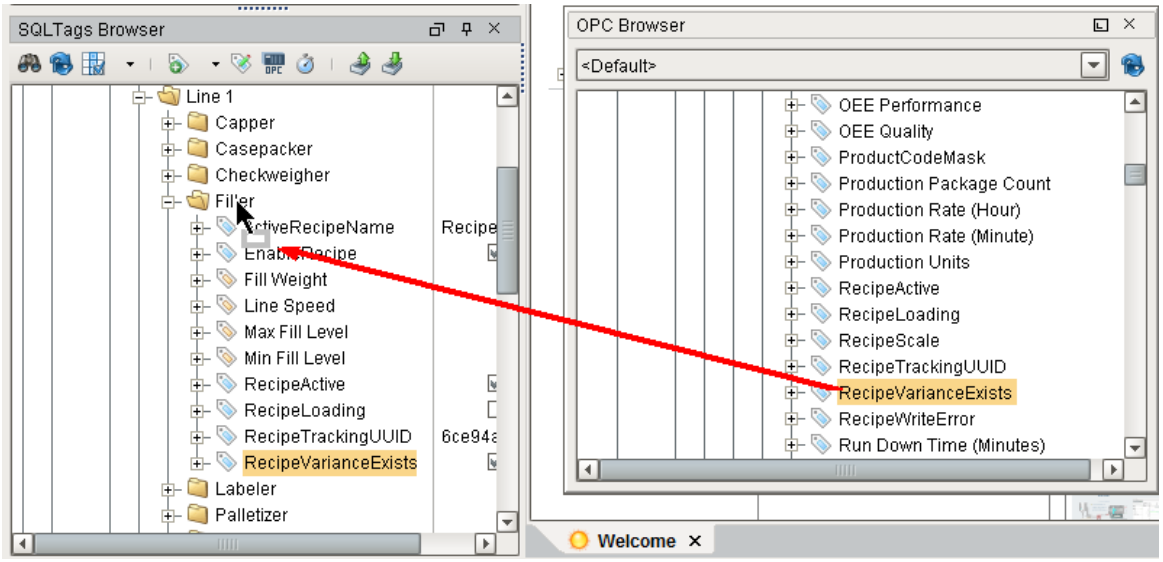
Recipe OPC Server

Production Model OPC Values

> **Info**
>
> When writing to production OPC values that are related to production model settings, the new value is not retained upon restarting. This is because production model settings are saved in the Ignition project and is only written to the project when done so in the designer.

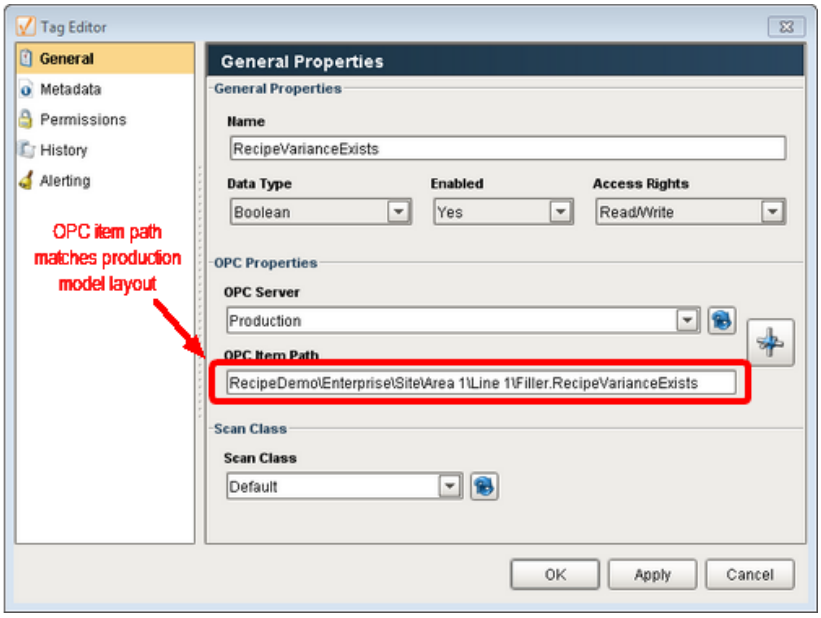
To use the production OPC values in your projects, Ignition tags have to be created. The easiest method to do this is drag the production OPC value to the SQLTag Browser. Once the tag has been created, it can be used to display status on screens, used in expression and any of the other tasks that can be done with any other Ignition tags. Most of the production OPC values are read only. For example, the RecipeVarianceExists value is determined by the live

production values compared to the values in the recipe (see Variance Monitoring for more details about the RecipeVarianceExists value). Because it is reflecting a status, it cannot be written to. However, others do allow writing a value to them.



Create Tag from Production OPC Value

It is extremely important to understand production OPC values have an OPC item path that matches the layout of the production model. In the image below, the RecipeVarianceExists tag is shown and includes the OPC Item Path of RecipeDemo\Enterprise\Site\Area 1\Line 1\Filler.RecipeVarianceExists. If tags have been previously created and the names are changed in the production model, then the OPC item path will also have to be changed.



Recipe Tag Editor

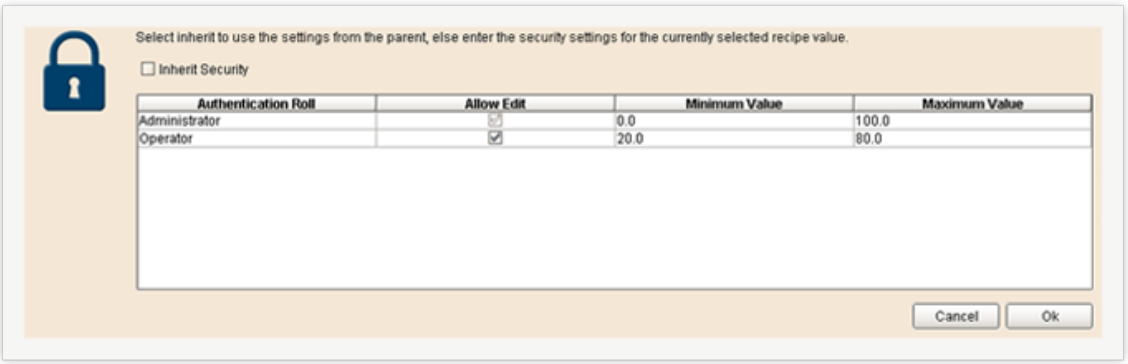
Tag Configuration

For example, if the enterprise name is changed from **Enterprise** to **My Big Company**, then the OPC item path for the tag named RecipeVarianceExists will have to change to RecipeDemo\My Big Company\Site\Area 1\Line 1\Filler.RecipeVarianceExists. For this reason, it is recommended to first work on laying out your production model and make sure the names of each of the production items are what you want before creating Ignition tags.

Track and Secure Recipes

Now you can extend the strong protection provided by Ignition's role-based security to your recipes. The Sepasoft Recipe / Changeover Module allows you to:

- Assign user roles
- Set permission for which roles can change which recipe values and by how much. For example, you can give the Administrator role permission to change a setting from 0–100, while limiting the Operator role's permission to change the setting from 20–80.



Track Recipe Changes

Whenever a change is made to a recipe setting, the details are recorded in the Sepasoft Recipe / Changeover Module's change log:

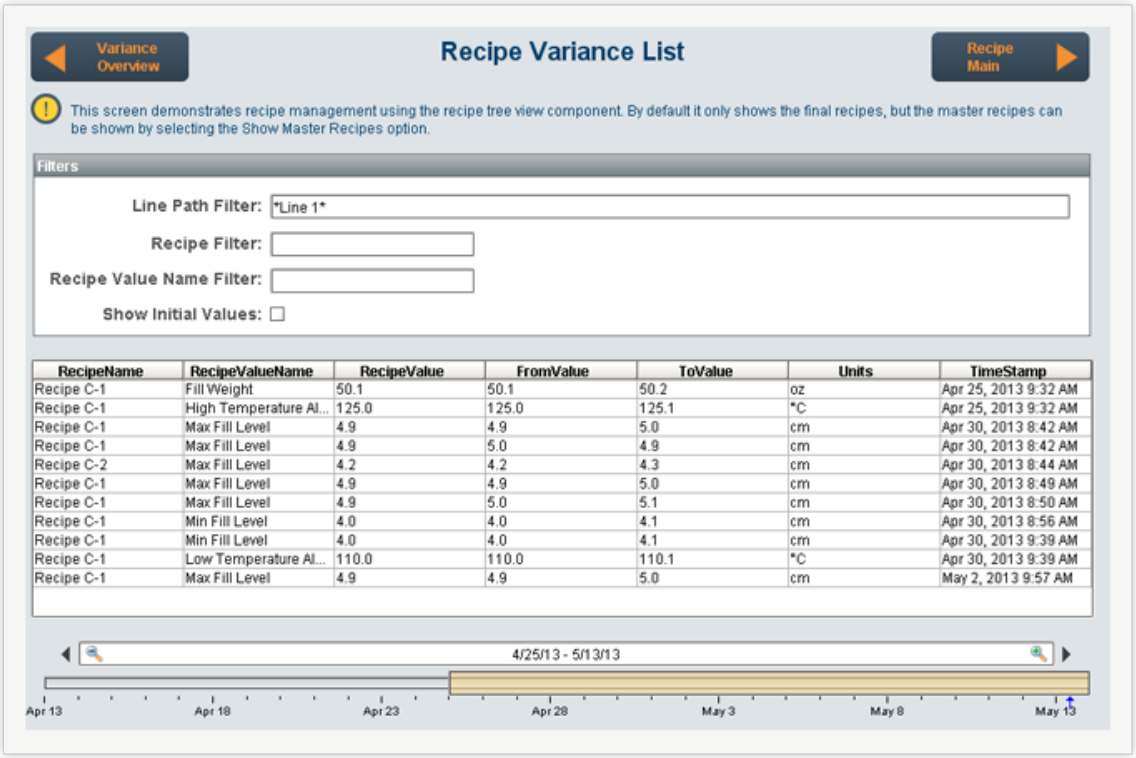
- Who made the change
- When the change was made
- What the previous setting was
- Why the change was made (you have the option to require an explanatory note)
- And more

The change log is valuable in normal production environments and is especially critical in industries with significant compliance requirements.

Monitor Recipes in Real Time

After a recipe is selected and once the initial recipe values are set, it is vital to monitor them for any variances to prevent quality issues, downtime or other production issues. During production, recipe values can be changed from systems outside of the recipe management system, such as an operator interface terminal local to a machine.

By monitoring the recipe values, the variance log in the Sepasoft Recipe / Changeover Module lets you detect variances in real time or review variances by production run or date range. These capabilities make it possible to identify the root causes of production issues early on.



The variance log also lets you define limits for which variances to record. The following types of thresholds can be defined by recipe value:

- Percentage +/- of recipe value
- Fixed +/- values from recipe value
- Fixed values
- Custom

Analyze Recipes

Use the Sepasoft Recipe / Changeover Module's built-in analysis tools to:

- Compare recipes
- Review recipe change logs
- Review production-run variances
- And more

When you add the Ignition Reporting Module, you can also create multi-page reports with the recipe analysis information, and more.

8.4.3 Recipe Module Configuration

There are three main aspects involved in setting up the Recipe Management Module.

1. Creating Recipe Tags
2. Configuring the Recipe Tab in the Production model
3. Creating Recipes

First we need to identify and setup the tags that will serve as a means to pass recipe setpoints down to the machine and also to monitor and track any variance. Next we need to make configuration settings in the production model that will affect how recipes and the recipe tags are managed. Finally we need to create recipes that will store the machine settings.

In this section, we will walk through the steps to add the management and control of machine settings using the Recipe Management Module.

Creating Recipe Tags

It is recommended to configure the tag database similar to the production model and specify the tag you are going to use as recipe set points.

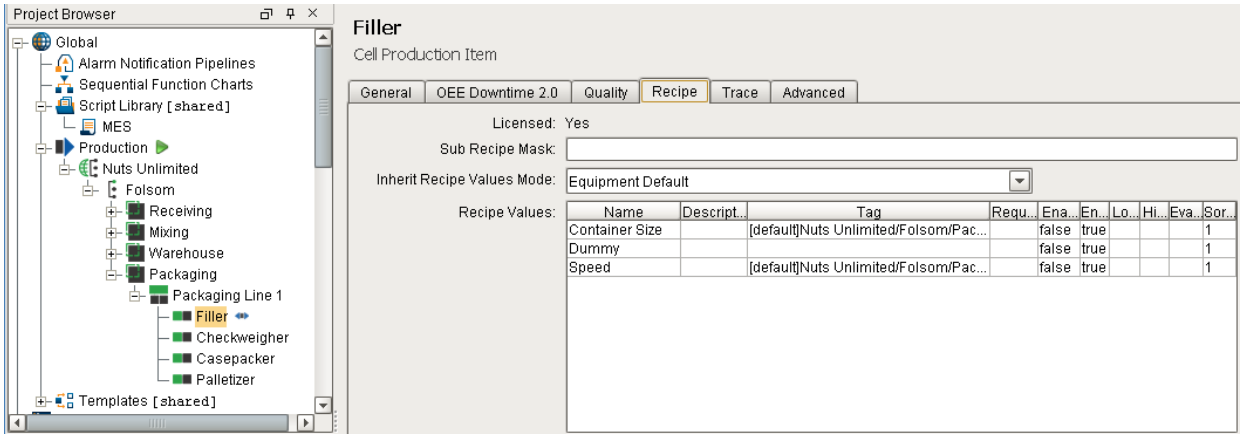
Recipe Configuration Settings

Recipe module configuration is done primarily in the **Recipe** tab for production items in the production model. Actual recipe values are stored in recipes that are created using the Recipe Editor components, but in the **Recipe** tab, the recipe value parameters for a machine that are available to recipes are created and bound to Ignition tags.

Recipe values are defined by production item. Each machine, process or other equipment will have settings that are unique. For example, a case-packer will not have the same settings as a mixer, so this is why recipe values are defined by production line, cell, cell group or location.

In This Section

- Recipe Values Settings



The following configuration is available on the Recipe tab:

- Recipe Value Propagation
- Recipe Value Inheritance
- Sub Recipe Mask
- Recipe Values Settings

Recipe Value Propagation

The Recipe module is very flexible in how machine settings can be stored and downloaded to a line. How you implement it will be dependent upon your recipe requirements.

- If you have a fairly simple line and all recipe setpoints will always be downloaded at once, you could simply create the Recipe Values settings and tags at the line level and be done.

- If you have a more complex line and recipe setpoints may be downloaded to cells on the line as a product moves through the process, then you may want to group Recipe Values settings and tags by cell or cell group.

Recipe Value entries are always propagated down to all child production items, If you have a recipe value setting that will be used for a number of cells on a line, for example **Line Speed**, you could define this setting at the line level and it would appear for all child production items. If this was a setting that was applicable to all production lines in an area, then it could be defined at the Area level. The name of the Recipe Value setting is propagated down, but the tag and recipe value is not by default.

Recipe Value Inheritance

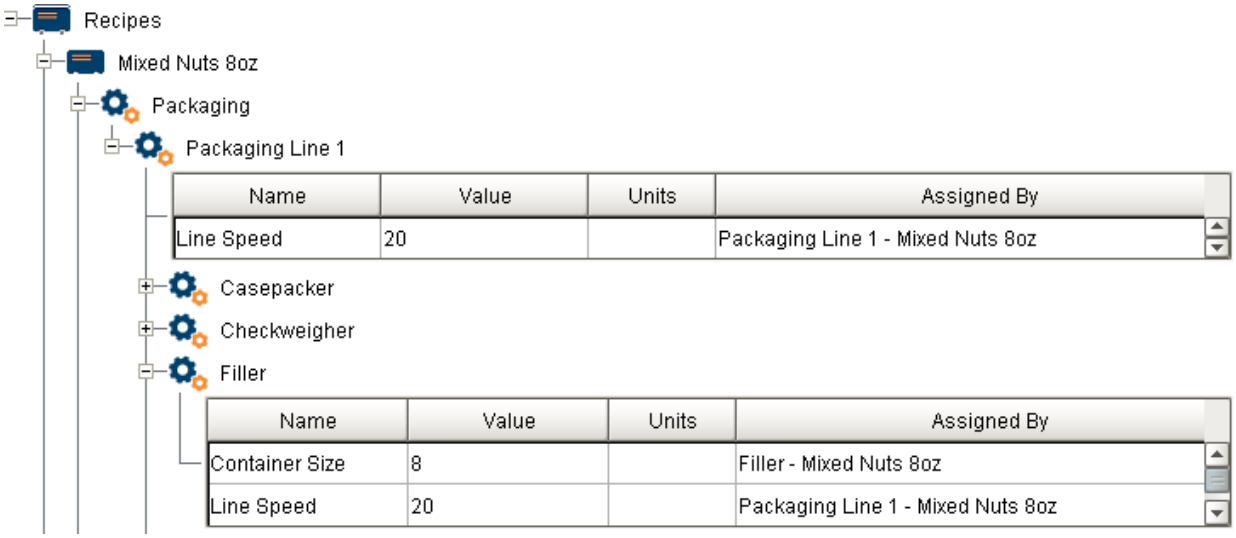
All Production items below the Area level can be configured to inherit their recipe values from the parent production item or to use their own recipe value based on the **Inherit Recipe Values Mode** setting.

Continuing the **Line Speed** example, after it was added to the Packaging line 1, it also became available for cells under that line. The Ignition tag associated with the recipe value at the line level is not propagated to the child recipe value and only the recipe values that have Ignition tags assigned to them will appear in the recipe editor. So, if a propagated recipe value is not relevant to the child production item, the recipe value tag property can be left blank.

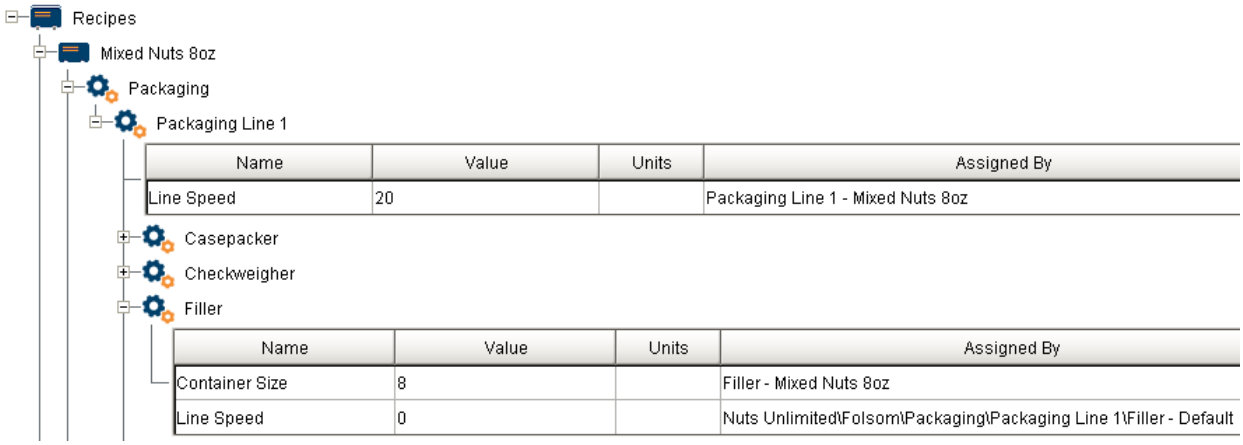
If the **Line Speed** recipe value at the filler cell is now given a tag path, it will show up in the recipe editor as a recipe parameter that can be set. The Inherit Recipe Values mode allows us to configure how the actual recipe value is obtained.

Inherit Recipe Values Mode

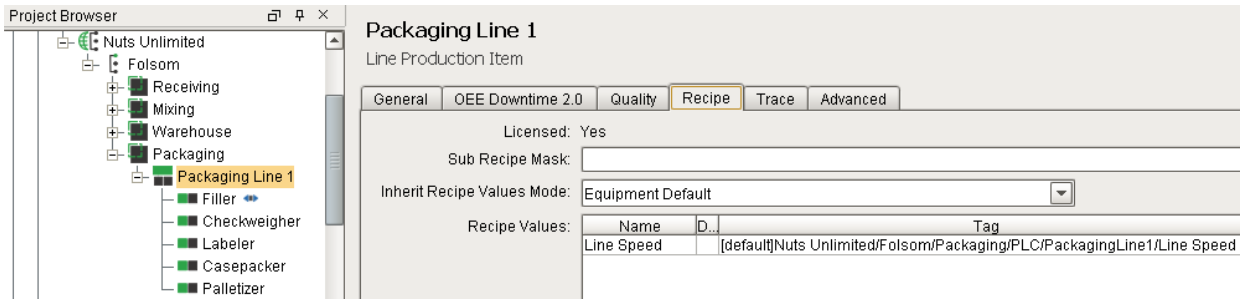
When set to **Parent Recipe**, the value of the Line Speed Recipe Values setting passed to the Filler recipe tag will be set to the value of the Line Speed setting set for the Line in the recipe editor.

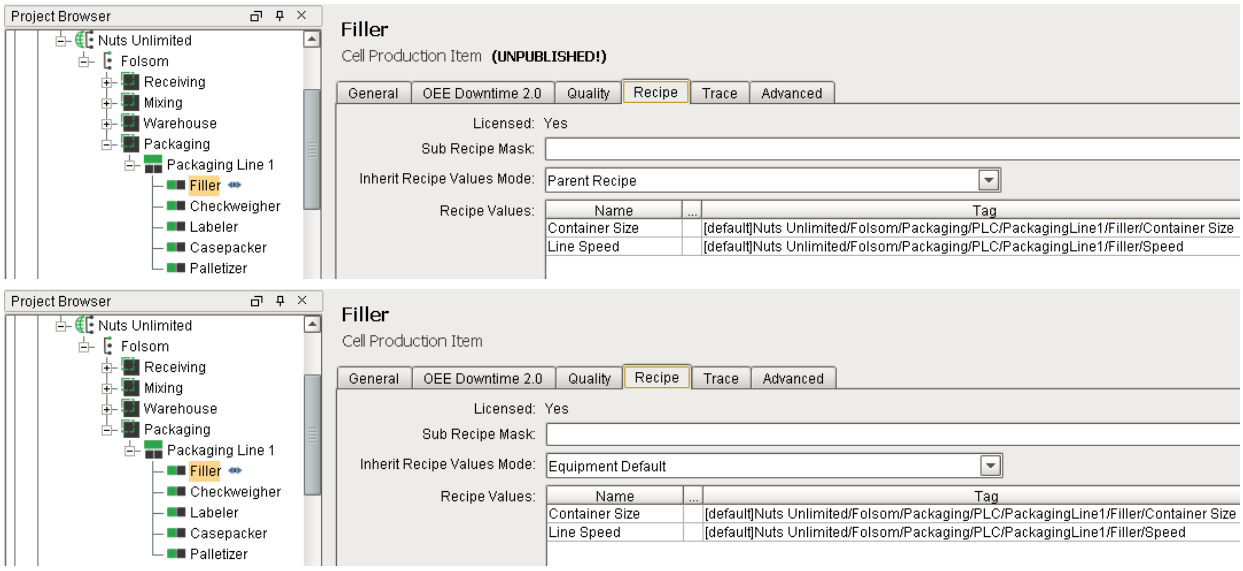


When set to **Equipment Default** the value of the Line Speed Recipe Values setting will be set to the value of the Line Speed setting set for the Filler in the recipe editor.



In either case, the value of the Line Speed parameter can be over-written using the Recipe Editor component by simply clicking on and assigning it a value.



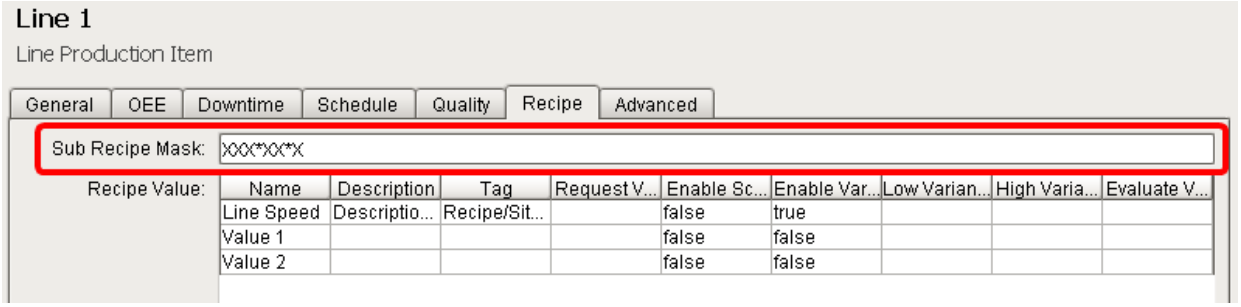


Sub Recipe Mask

In some situations, a limited number of digits in the product code can be used to specify the recipe to use for a machine. This is accomplished by setting a sub recipe mask. See Sub Recipes for more information.

To set the Sub Recipe Mask for a line, cell, cell group or location, first select the desired production item, then select the Recipe tab and enter the new Sub Recipe Mask value.

The Sub Recipe Mask must be set to be able to use the sub recipe feature and only applies to line, cell, cell group or location type production items. If the sub recipe feature is not being used for the production item, leave it blank. The sub recipe feature can be used on a production item by production item basis. This means that standard recipe and sub recipe functionality can be mixed. For example, you may have a tape machine that recipe values only change based on the case size. If there are only two different case sizes and there is a digit in the product code that specifies the case size, then sub recipes can be used for the tape machine. All other machines can use the normal recipe functionality.



Recipe Values Settings

There are a number of settings that can be defined for each Recipe Value by right-clicking on the Recipe Values entry in the table and selecting **Edit** or **New** if you are going to create a new Recipe Value. Right-click and select **Delete** to delete a Recipe Value.

We'll deal with each setting in the following section Recipe Values Settings.

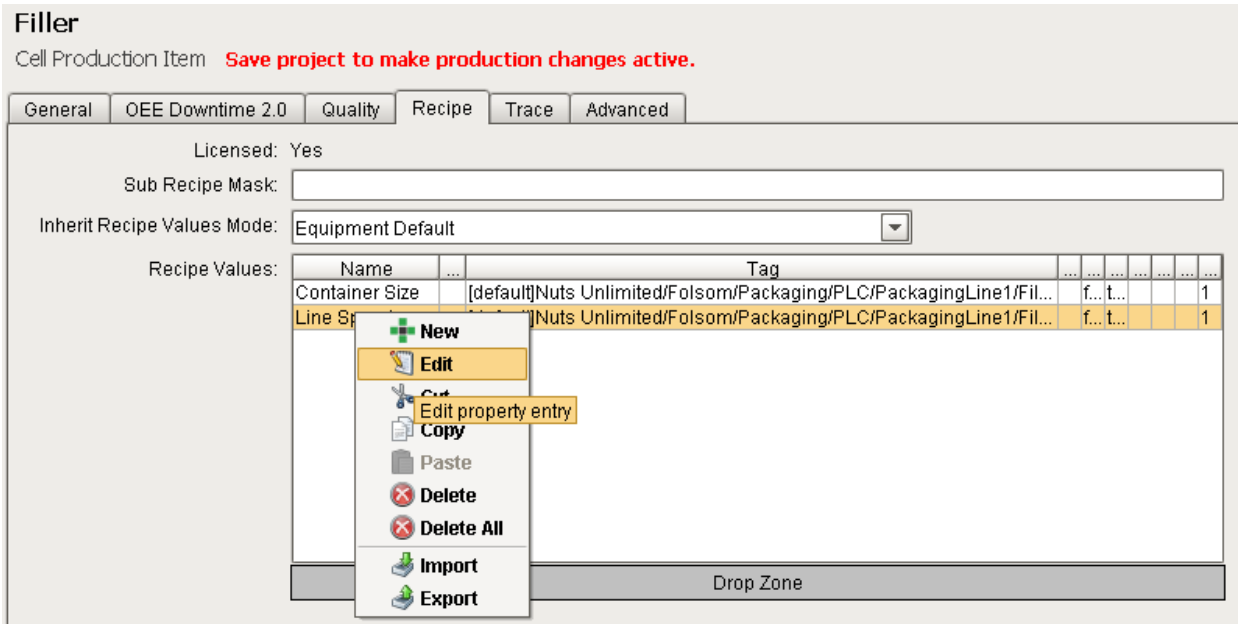
Importing and Exporting Recipe Values Settings

To import recipe value configuration entries, right-click anywhere on the recipe values table and select the Import menu item. A dialog box will appear to allow selection of a comma separated values (csv) formatted file.

The first line of the file must at least contain the property names separated by commas. If additional names exist, they will be ignored. The property names can be in any order. Below is a sample csv file showing multiple recipe value configuration entries.

```
ValueName,ValueDescription,ValueSQLTag,ValueCalcScript,AllowScaling,
ValueMonitorEnabled,ValueMonitorLow,ValueMonitorHigh,ValueMonitorScript

"Line Speed","Description for Line Speed","Recipe/Site/Area 1/Line 1/Line Speed","","
false","true","","","",""

"Value 1","","","","","false","false","","","",""

"Value 2","","","","","false","false","","","",""
```

To export recipe value configuration entries, right-click anywhere on the table containing recipe value configuration entries and select the **Export** menu item.



Recipe Values Settings

In the Recipe Values dialog box, the following settings can be applied:

- Add a Recipe Value name
- Add a Recipe Value description field
- Add a tag path to the tag that will hold the recipe value
- Add a 'Request Value' script
- Enable value scaling
- Enable variance logging and add variance thresholds
- Add an 'Evaluate Variance' script
- Define a Sort Order

Name

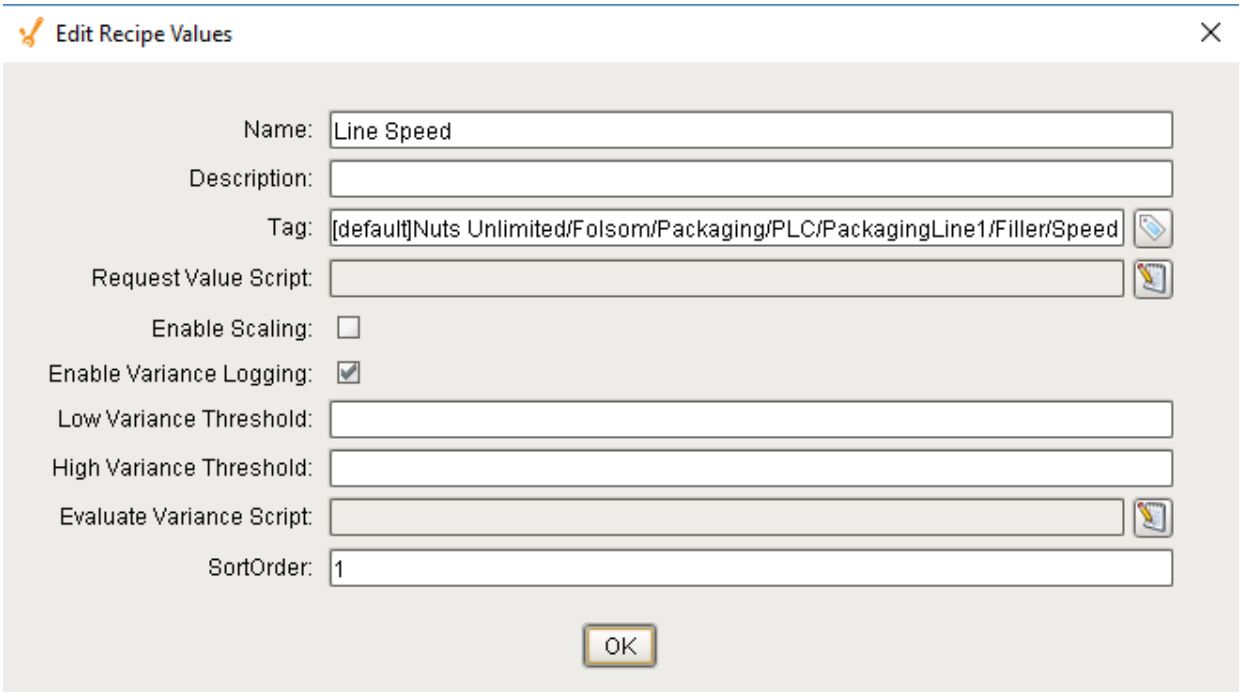
The required name is used to reference the recipe value. The name must be unique and must not exist in any of the child production items of the production item that the recipe value is being added to. The reason for this is that recipe values are propagated down to all of the children, and if the name is the same, a conflict will occur. Also, some characters are not allowed in recipe value names.

Description

The recipe value description is used to further describe the recipe value. It appears in the recipe editor component, analysis, reports, and etc.

Tag

This is the path to the Ignition tag that is associated with this recipe value. If a recipe value is added but no tag is assigned, it will not appear in the recipe editor, and values will not be used when recipes are selected.



Request Value Script

Script can be added to calculate or obtain a value to return for a recipe value anytime a recipe is selected for a production item. This provides flexibility to do just about anything in place of returning the value stored in the recipe management system. See Request Value Script section for more information.

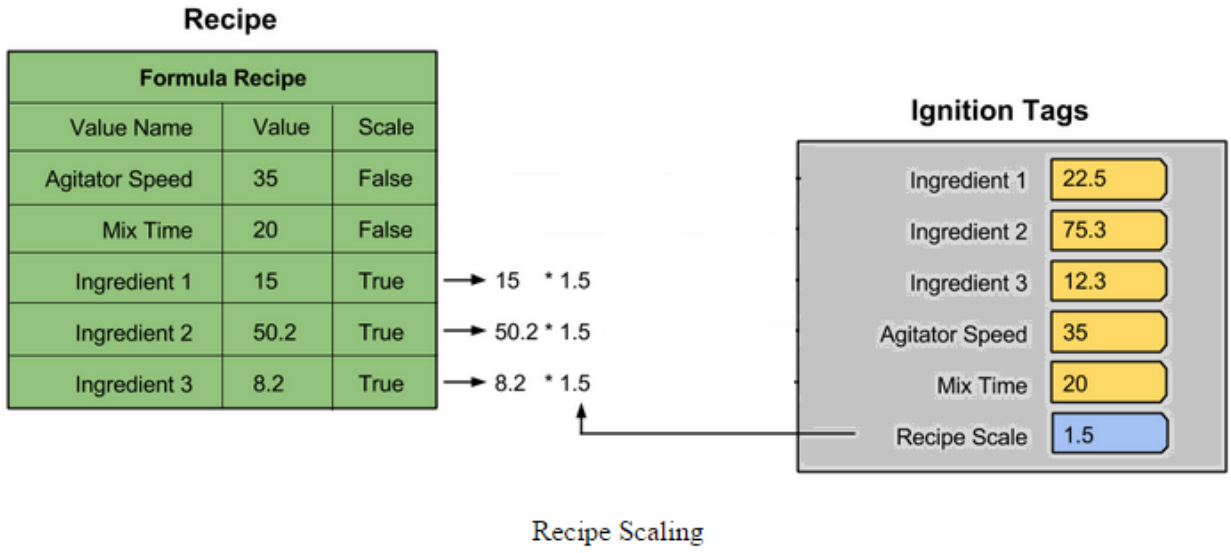
Enable Scaling

If this option is checked, the recipe value will be scaled. The recipe value is retrieved out of the recipe management system and then scaled by the value of the recipe scale tag for the production item.

When using recipes for batch or other processes that can change based on the amount that is produced, recipe scaling will adjust recipe values based on a recipe scale value. In the recipe value configuration, there is an Enable Scaling setting that can be selected. If the Enable Scaling setting is selected for a recipe value, then whenever a recipe is selected for a production line, cell, cell group or location, the value from the recipe will be scaled by the value in the RecipeScale tag as shown in the image below.

Enabling recipe scaling is done for each individual recipe value. This supports scaling some of the recipe values while not scaling others as might be the case in the example shown below. By default, each production item's Enable Scaling setting is false and must be selected before the RecipeScale value will be applied.

The RecipeScale is a production OPC item that exists for each production line, cell, cell group or location. By default, the RecipeScale is 1.0 and recipe values will not change when recipes are selected. When selecting a recipe for a line, all of the cells, cell groups and locations beneath the line will also be set to the same recipe provided they are enabled. Also, each cell, cell group and location RecipeScale value will be set to match that of the line. This enables simple recipe selection for a line without the tedious task of selecting each machine underneath the line.



Recipe Scaling

Enable Variance Logging

If this option is checked, then the tag will be monitored for changes after a recipe is selected for a production item. If the value changes more than the window defined in the Low Variance Threshold and High Variance Threshold, the variance will be logged to the database, and the Recipe Variances Exists OPC tag for the production item will be set to true. This prevents values that are known to vary within an allowable range from being logged to the database and causing the Recipe Variances Exists tag from being set.

If this option is not checked, then the value can change and it will not be logged to the database. Also, the Recipe Variances Exists tag will not be set as a result of this recipe value.

See Variance Tracking for more information.

Low Variance Threshold

The Low Variance Threshold setting is used to define the lower limit before recipe variances are triggered for this recipe value. The variance threshold can be defined as a percentage of the recipe value or a fixed amount.

See Variance Tracking for more information.

High Variance Threshold

The High Variance Threshold setting is used to define the upper limit before recipe variances are triggered for this recipe value. The variance threshold can be defined as a percentage of the recipe value or a fixed amount.

See Variance Tracking for more information.

Evaluate Variance Script

Script can be used instead of using the Low Variance Threshold and High Variance Threshold settings to determine if the recipe value is outside of an allowable range. When the Recipe Values tag value changes, the variance state is evaluated using the Low Variance Threshold and High Variance Threshold settings. Then, if an Evaluate Variance Script has been entered for the recipe value, the script will be executed, and the state can be changed. See Variance Tracking and Evaluate Variance Script for more information.

Sort Order

The order in which Recipe Values appear in the Recipe editor can be modified by setting the Sort Order value. This is an integer value with higher values appearing first in the Recipe Editor. The default value is 1.

Recipe OPC Production Tags

The recipe module creates a number of OPC Production tags that provide recipe status that can be used in your application.

For more information, refer to the OPC Production Server Tag Reference help in the Appendix: reference guide.

8.4.4 Creating Recipes

Recipe values are configured for each production item. Each machine, process or other equipment has settings that are unique. For example, a case packer will not have the same settings as a mixer, this is why recipe values are defined by production line, cell, cell group or location.

Recipe Editor Overview

Recipe Editor - Default Values and Security

Recipe Editor - Adding Recipes

Editing Recipes

There are multiple methods that can be used to change the values of a recipe. Depending on the functionality that you are looking for, recipe values can be changed using the recipe editor, importing or by scripting.

Recipe Editor

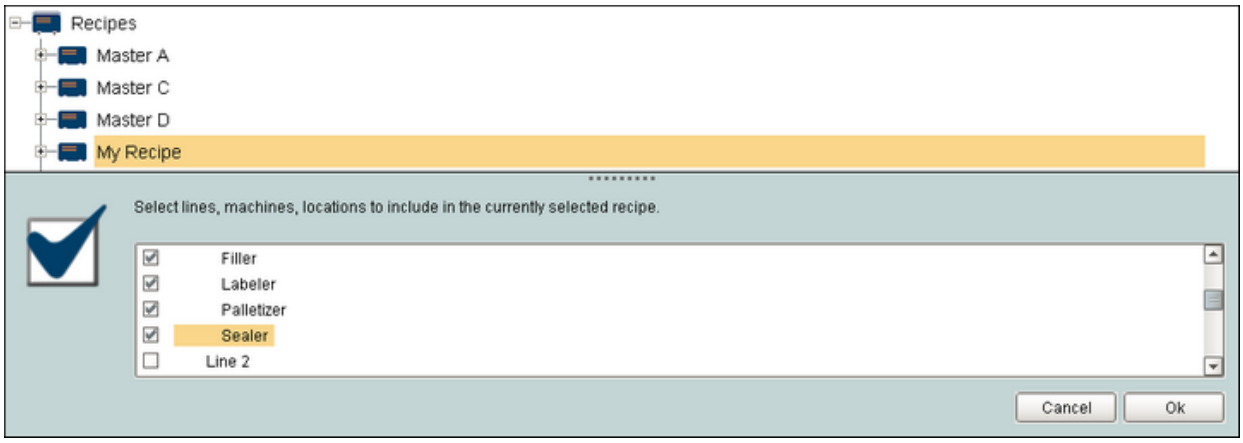
The recipe editor component provides a visual and interactive method to allow end users to manage recipes. It handles all of the details and is as easy as adding the component to any Ignition window. It also provides the ability to manage sub-product codes, recipe value security, master recipes and adding MES production items to recipes.

To add a new recipe, right click on the root Recipes item in the recipe editor and select Add Recipe menu item. The new recipe will be added and will be ready to enter the name of the new recipe. Commonly, the name of the recipe will be the same as a product code, but it does not have to be. It can represent a mode of the machine such as Cleaning Mode.

Add Recipe

Type in the name of the new recipe which for this example it is My Recipe. Next, right click on the new My Recipe and click on the Select Production Items menu item.

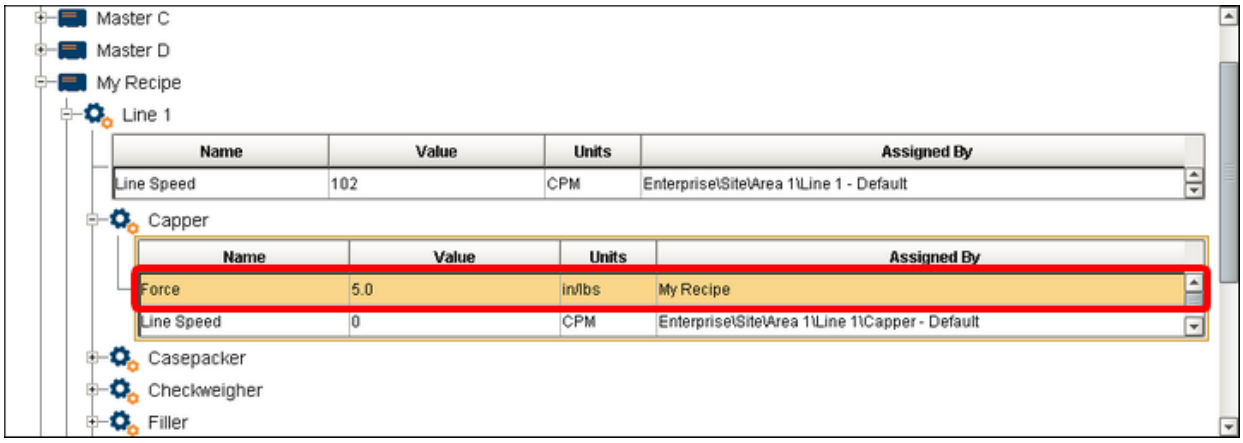
> **Info**
>
> Please note, that you must first add production items in the designer before they appear as options to be added to a menu. Because not all machinery is used in every recipe, this step is used so that only the machinery that is appropriate for a recipe appears in the recipe editor and recipe selector components. For this example, Line 1 and all of the cells (machines) beneath it are added to the recipe.



Recipe Select Prod Items

Select Production Items to Add to Recipe

After clicking the OK button, expand the production item to view and edit the recipe values. Notice the Assigned By column. When the new recipe was first added, all of the recipe values show as assigned by Enterprise\Site\Area 1\Line 1\Capper - Default for the capper. This is because the initial values of a new recipe are inherited from the default values for the capper production item. See Default Values for more information. When a recipe value is changed, the assigned by changes to My Recipe. This is because the value is no longer that from the default values and is now from the recipe. In simpler terms, it tells you where the value has been changed in the inheritance tree.



Recipe Edit Value

Edit Recipe Values

My Recipe will now appear in the Recipe Selector List component and can be selected for any of the production items that were added to the recipe. But, My Recipe can also be made into a master recipe simply by adding descendant recipes to it. See Master Recipes for more information. This is done by right clicking on the Descendants item beneath the My Recipe recipe in the recipe editor, and clicking on the Add Recipe menu item. Type in the name of the new recipe which for this example it is My Recipe 1. There is no limit to the number of

descendants recipes you can add to a master recipe. There is also no limit to the number of levels deep of master recipes. After My Recipe 1 is added to My Recipe, My Recipe will no longer show as an option in the recipe selection list component but My Recipe 1 will. In general, if a recipe has descendants, it becomes a master recipe and will no longer show in the recipe selection list component. Only final recipes with no descendant will show in the recipe selection list component.

However, master recipes can be selected for production items by using script functions.

Import / Export

Recipes values can be imported and exported into the recipe management system. Note that only the actual value of the recipe value item can be imported and not the recipe value definition. Recipe value definitions can be imported in the designer. See Recipe Value Import / Export for more information. This is because the recipe value definitions are tightly tied to production items (equipment) and tags, both of which cannot be created in the client.

To export the recipe values in the client, right click on a line, cell, cell group or location underneath the line and select the export menu item. A file chooser dialog will appear to select or enter a file name to export to. The file format is a comma separated values (CSV) and contains the following columns:

Recipe_NameValue_Name

Item_PathDescriptionUnits
Data_TypeFormatRecipe_ValueAssigned_By

To import recipe values in the client, right click on a line, cell, cell group or location underneath the line and select the import menu item. A file chooser dialog will appear to select or enter a file name to import to. The file format is a comma separated values (CSV) and must contain the following columns:

Recipe_NameValue_NameItem_Path
Recipe_Value

All other columns will be ignored during the import. Also, all values must be surrounded with quotes including the recipe value. During importing, the recipe value will be converted to the appropriate data type that the recipe value is defined as, which is based on the tag it is associated.

Recipe values can be imported for multiple recipes and production items combinations in one import operation as defined with the Recipe_Name and Item_Path columns of the CSV file. This supports bulk import operations instead of only being limited to one recipe at a time.

Recipe values can also be imported and exported using script either at the client or in the gateway. See system.recipe.exportRecipe and system.recipe.importRecipe script functions for more information. The following is an example statement that will import recipe values on the

gateway. The project name is required because recipe values are managed by project. The csvData parameter is a string of csv data that can be read in from a file, web service, etc. And last, the note is what will show in the recipe change log.

system.recipe.importRecipe("RecipeProject", csvData, note)

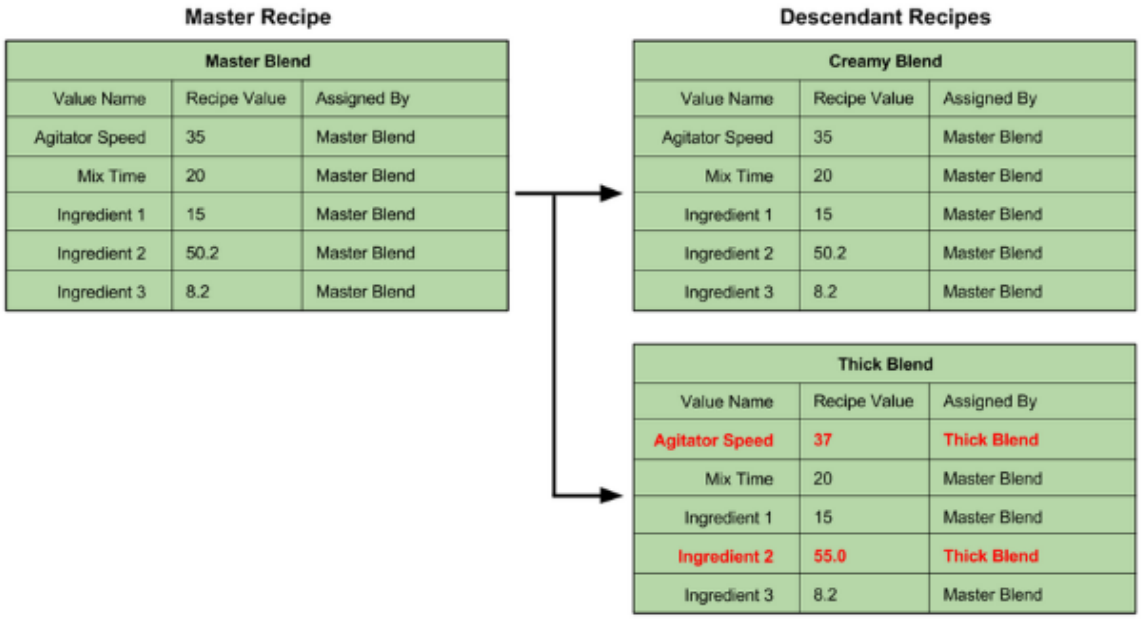
This functionality supports reading recipe values from ERP or other systems that are currently being used to manage recipes. Once the recipe values are in Ignition they can be selected, monitored for variances, analysis, etc.

Script

In addition to importing and exporting recipe values there are scripts to add recipes, rename recipes, delete recipes and much more. See Client / Gateway Scripts for documentation of all script functions. Because the built-in functionality will not fit the requirements in every situation, the scripting functions provide the built-in functionality to be extended to accommodate the requirements.

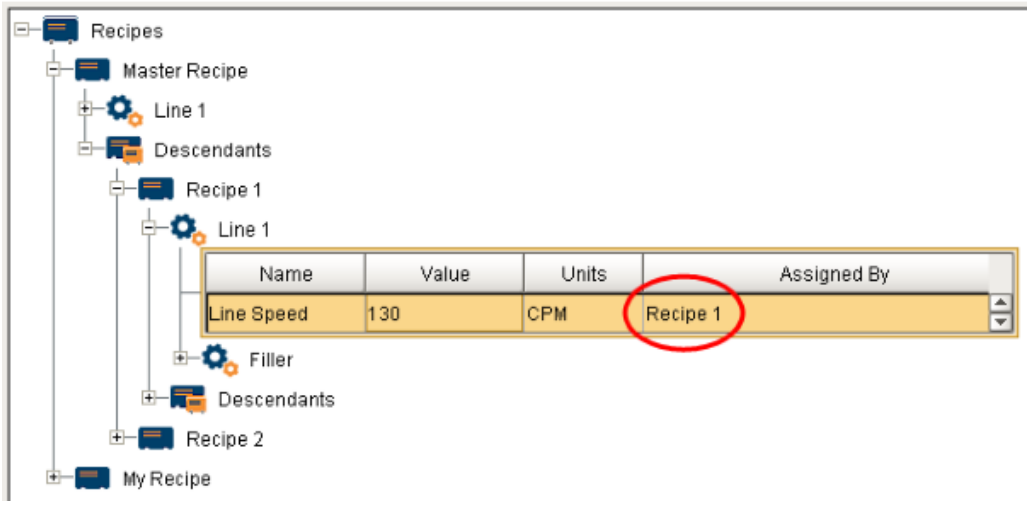
Recipe Editor - Master Recipes and Descendants

Making a change to a recipe value that is used in numerous recipes is a daunting task and is prone to mistakes. To address this problem the Recipe / Changeover Module uses master recipes. The image below shows two recipes that are derived from, or descendents of, the Master Blend recipe. When the descendant recipe is added, all recipe values will be inherited from the master recipe. When a value is changed in a descendant recipe, it will override the value from the master recipe with the new value as shown in the image below for the Agitator Speed and Ingredient 2 recipe values.

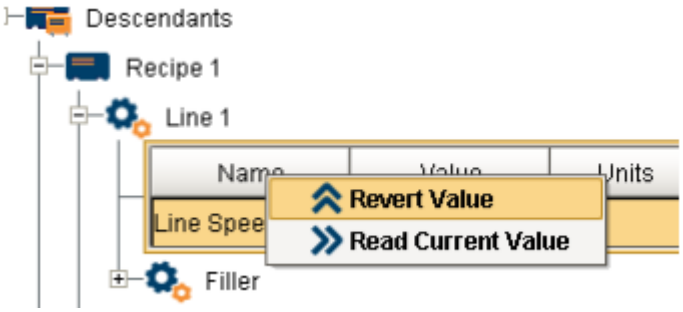


It is very easy to add descendant recipes. Create a new recipe called **Master Recipe**. Make sure to select the production items **Line 1** and **Filler**. Set the recipe values for the master recipe.
To add a descendent simply right click on the **Descendents** folder and select **Add Recipe**.

Give the descendent recipe a name like **Recipe 1**. Configure a second recipe called **Recipe 2**. Now we have 2 descendants. You can expand the descendants to specify the recipe values. If you modify a value in the descendant it will set the **Assigned By** to the descendant like **Recipe 1**.



At any time you can right click on a recipe value and revert the value. This will set the value back to the parent.



Recipe Editor - Sub Recipes

Recipe Editor - Notes

Recipe Editor - Filtering

The Recipe Editor component provides a number of properties that allow for the filtering of recipes that will be shown in the Recipe Editor. These filters can be used to limit who has access to view which recipes or for reducing the number of recipes shown in the Component.

The filters available are dependent upon the component being used.

Available filters are...

Item Path Filter

Recipe Name

Recipe Value Name Filter

Recipe State Filter

Recipe Group Filter

Refer Recipe Components for more information on available component filter options.

Recipe Editor - Allowing Certain Functions

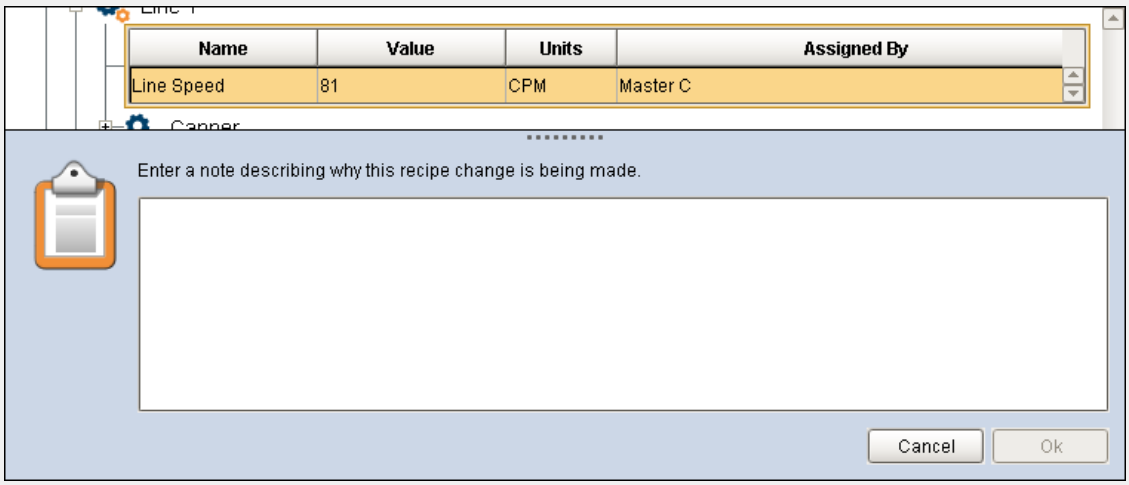
Recipe Editor Table

Recipe Editor Examples

Customizers

Examples

Based on the setting of the Require Note property, notes are required any time changes are made to a recipe, sub recipe or default values. The note panels is shown below and the appearance is defined by several properties. The Popup Panel Font property determines the font of the text, the Note Panel Icon Path property determines the image on the upper left hand corner and the Note Background Color property determines the background color. This is just an example of the many properties that change the appearance of the Recipe Editor component.



Recipe Editor Note

Recipe Editor Note

This component does not have any custom properties.

It has the capability for end users to do the following recipe related tasks:

- Manage recipe value security.
- Manage sub product code recipes.
- Manage default machine values.
- Manage master recipes.
- Manage machines that a recipe can be run.

Based on the setting of the properties of the Recipe Editor component, more or less detail can be shown. This provides a method of displaying the correct amount of information depending on the logged in user's authentication roles. For example, the image below is very clean only showing limited recipes. This mode allows changing of recipe values for final recipes (not in master recipes). See Master Recipes for more information.

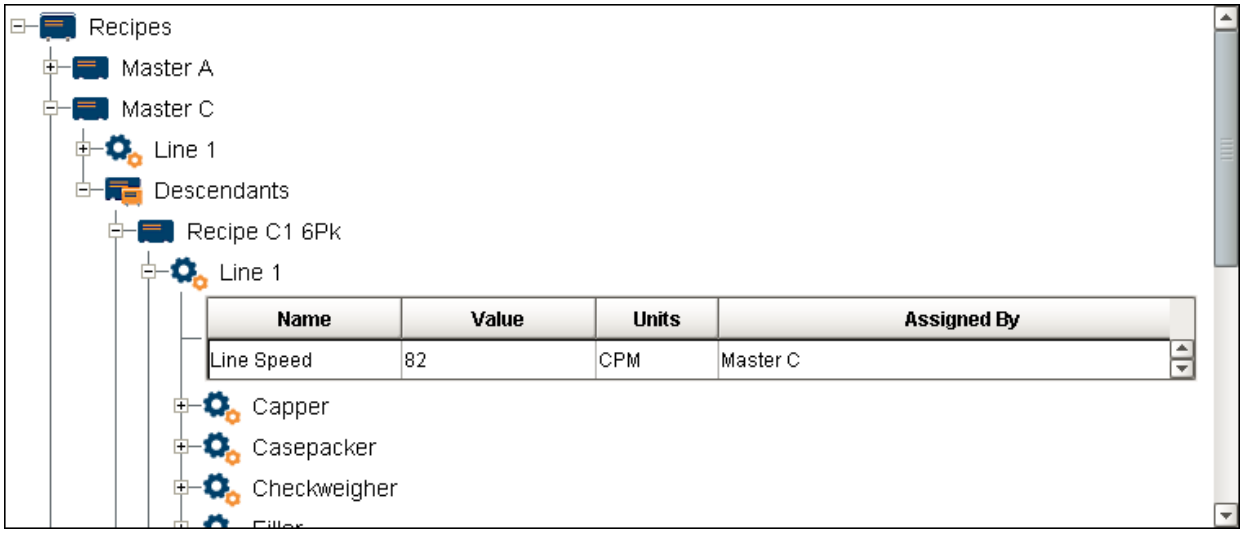
Recipe Editor Simple

Basic Recipe Editor

Where as this image demonstrates a lot of recipe information with many more options, providing much more configuration of recipes. The Show Master Recipes property will determine if master recipes are shown. See Master Recipes for more information.

New master recipes can be added by right clicking on the root Recipes node and selecting the Add Recipe menu item.

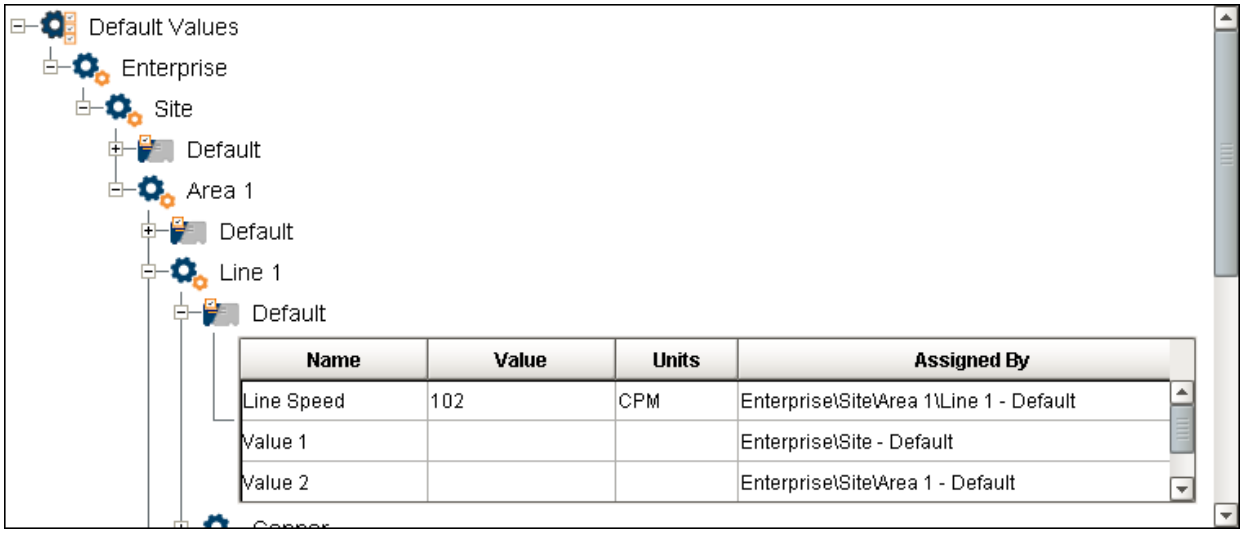
New descendant recipes can be added by right clicking on the Descendants node and selecting the Add Recipe menu item. Existing descendant recipes can be renamed, removed or etc. by right clicking on the descendant recipe node and selecting the desired menu item.



Recipe Editor Complex

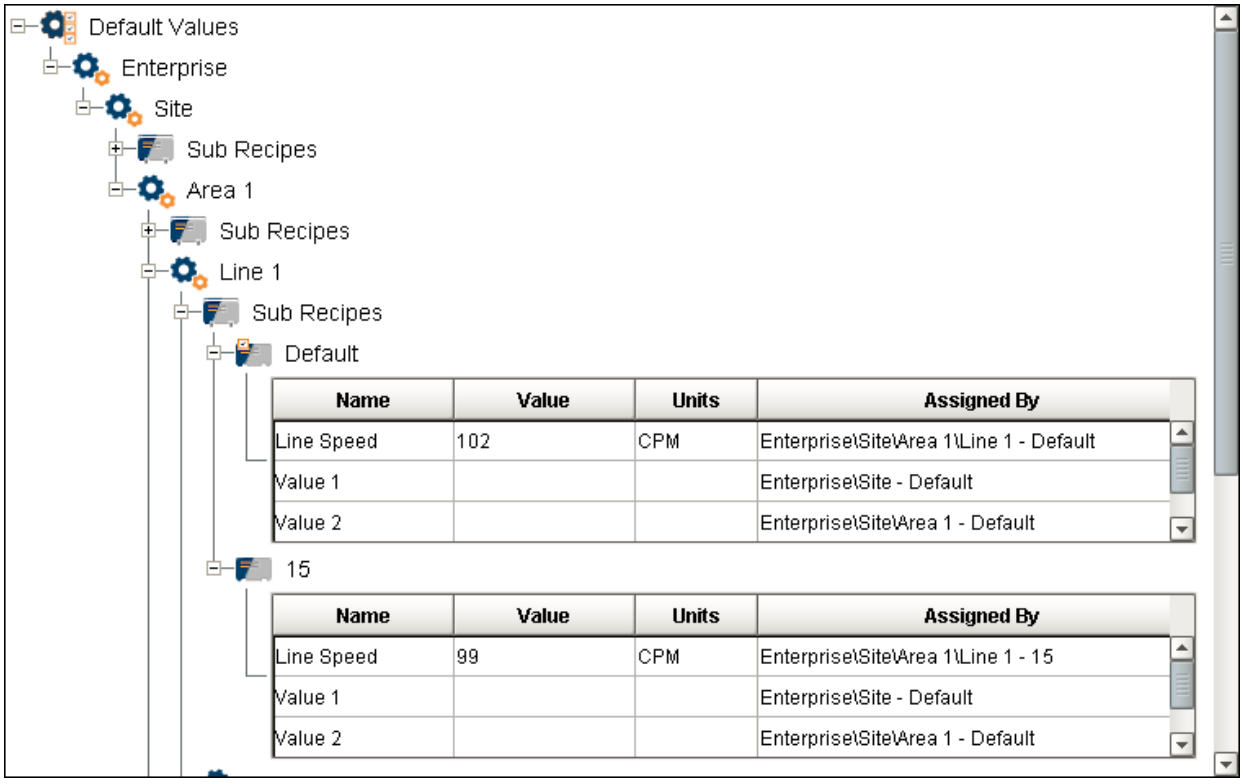
Full Recipe Editor

In addition to editing recipes, default values for machines (production lines, cells, cell groups and locations) can be managed as shown in the image below. The Show Item Defaults property determines if the default values root item is shown in the recipe editor. See Default Values for more information.



Default Value Editor

Sub recipes can also be managed by setting both the Show Item Defaults and Show Sub Recipes properties to true. See Sub Recipes for more information. New sub recipes can be added by right clicking on the Sub Recipes node and selecting Add Sub Recipe menu item. Sub recipes can also be removed, renamed or etc. by right clicking on the node of a sub recipe and selecting the desired menu item. The Default sub recipe is always shown and cannot be renamed or deleted. It is reserved for holding the default values for a machine.



Recipe Editor Sub Recipes

8.4.5 Loading Recipes

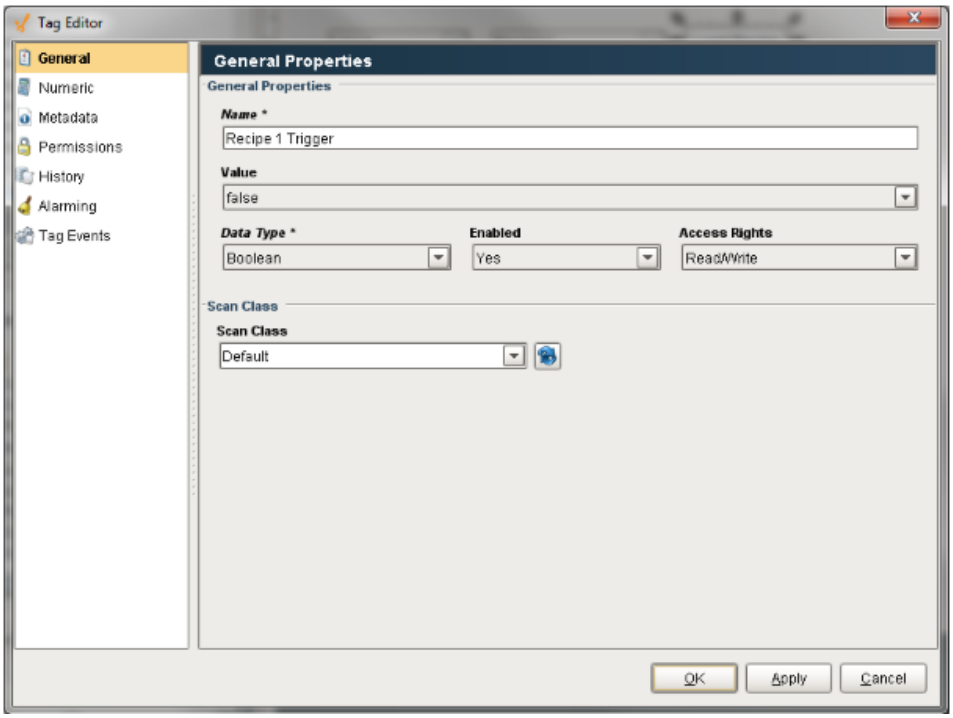
There is more than one way to load or add new recipe values to a production item. You can use the menu commands and popup windows or you can drag and drop one or more tags from the Tags Browser to the recipe value table.

Manual - Component

Manual - Scripting

Automatic - Scripting

We can also load recipes automatically from a trigger in the PLC, SFC chart, and many other places. Let's show how to load a recipe to the PLC when a trigger changes in the PLC. In the tag browser, create a memory tag in the **Line 1 > PLC** folder called **Recipe 1 Trigger** that is a boolean with a default value of false.



Lastly, select the **Tag Events** tab so we can setup a script that will run automatically and load **Recipe 1** when the tag goes to 1.

Select the **Value Changed** script and enter in the following script:

```
#if not initialChange and previousValue.value != currentValue.
value and currentValue.value:
linePath = "[global]\My Enterprise\Site 1\Packaging\Line 1"
recipe = "Recipe 1"
system.recipe.setItemRecipe("RecipeDemo", linePath, recipe, 1)
```

Set the trigger to 1 and watch the recipe get loaded down automatically.

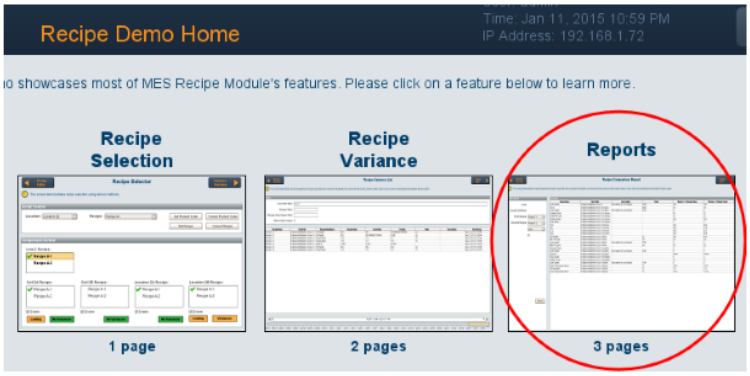
8.4.6 Analysis and Reports

Use the built-in analysis tools to compare two or more recipes, review recipe change logs, and review production-run variances. You can create multi-page reports with the recipe analysis information and more.

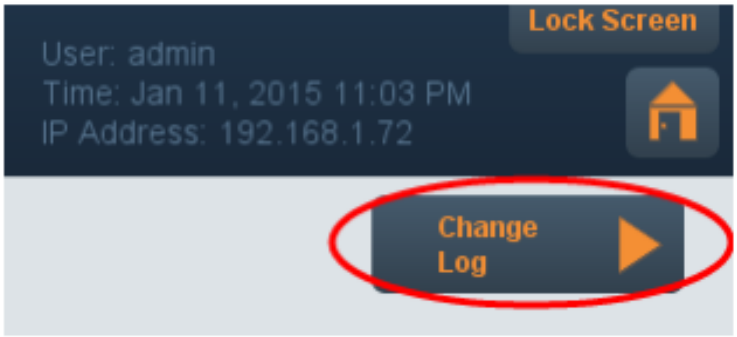
Change Log

The Recipe demo project provides a PDF report to see the change log history. Launch the **RecipeDemo** project as a runtime.

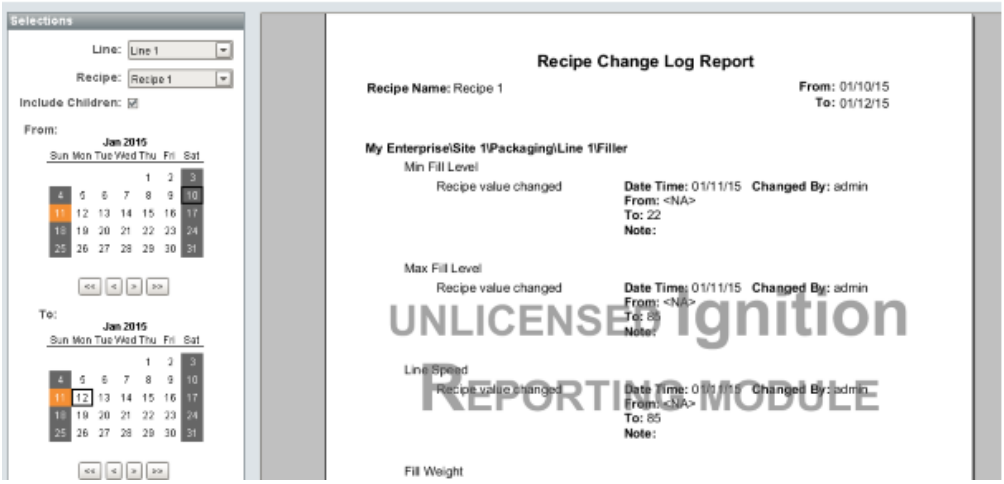
Once the client is launched, log in and select the **Reports** image.



Click on the **Change Log** button in the top right.



To use the **Recipe Change Log Report** simply select **Line 1** in the line dropdown box, choose Recipe 1 as the recipe. Choose a start and end date. Press **Refresh** to load the report.

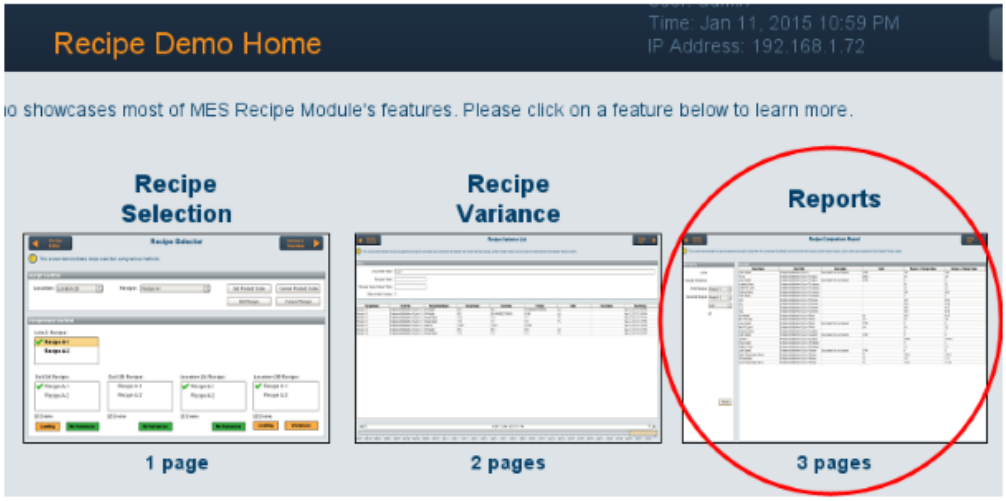


If you want to see how the window was configured take a look at the **Reports > Change Log** window in the designer. It makes use of the Anaylsis Controller (the same component we used in the OEE / Downtime module) and the Recipe Change Log provider.

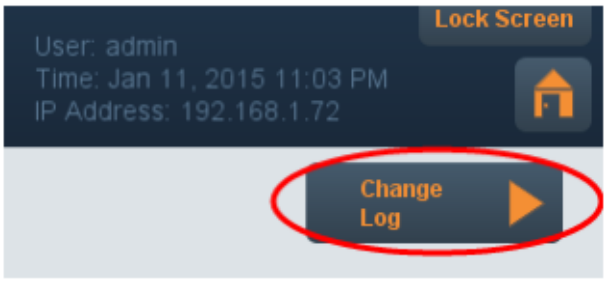
Variance Log

The Recipe demo project provides a PDF report to see the change log history. Launch the **RecipeDemo** project as a runtime.

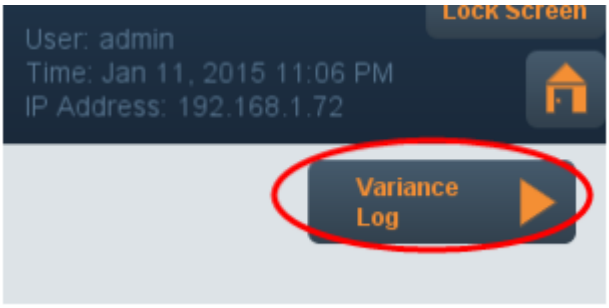
Once the client is launched, log in and select the **Reports** image.



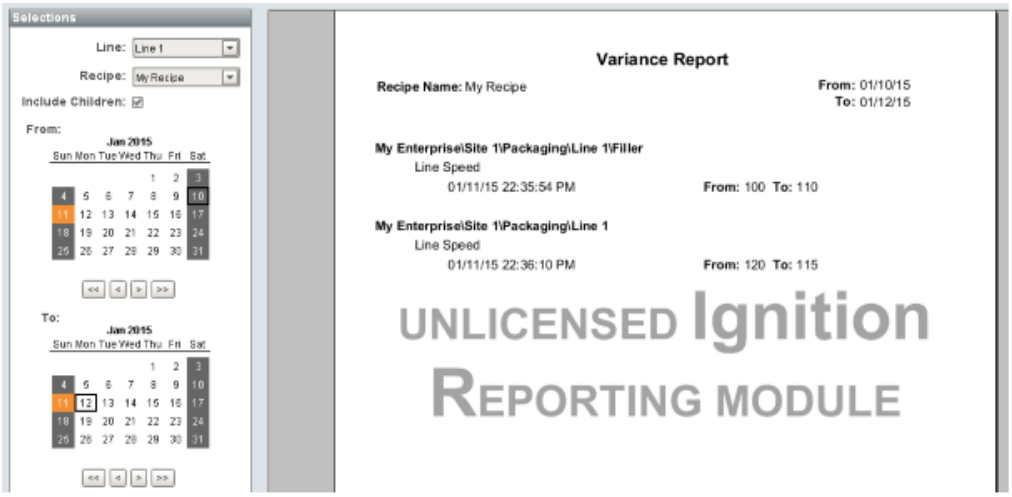
Click on the **Change Log** button in the top right.



Click on the **Variance Log** button in the top right.



To use the **Recipe Variance Report** simply select **Line 1** in the line dropdown box, choose Recipe 1 as the recipe. Choose a start and end date. Press **Refresh** to load the report.



If you want to see how the window was configured take a look at the **Reports > Variance** window in the designer. It makes use of the Anaylsis Controller (the same component we used in the OEE / Downtime module) and the Recipe Variance provider.

8.4.7 Recipe Values

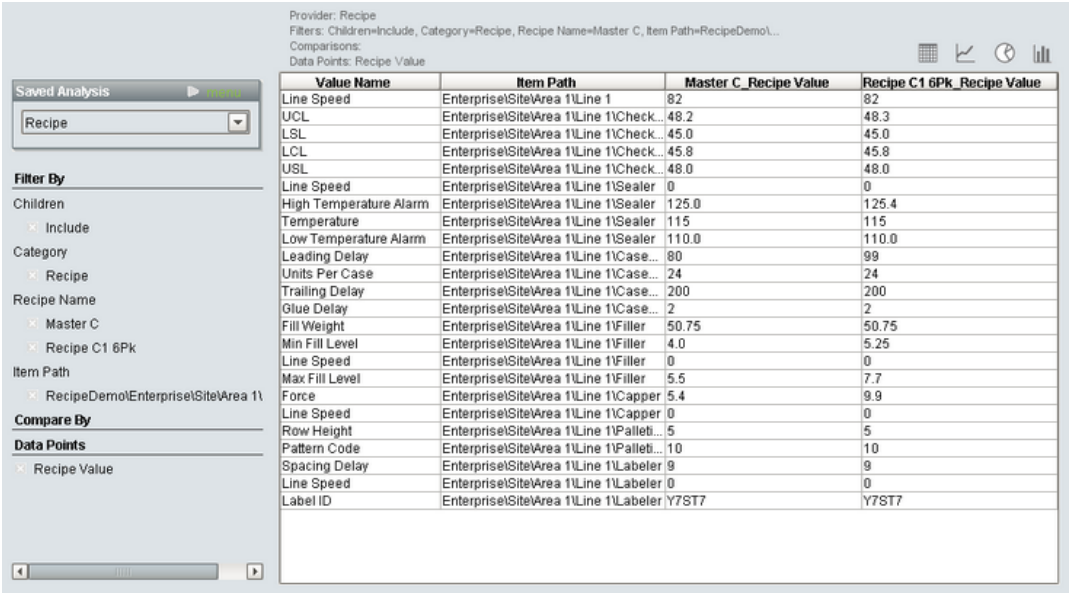
Recipe values are defined by production item. Each machine, process or other equipment will have settings that are unique. For example, a case-packer will not have the same settings as a mixer, so this is why recipe values are defined by production line, cell, cell group or location.

See Recipe Types for more information on how recipe values work. The following sections detail how to add, edit, delete, export and import recipe values for a production item.

8.4.8 Analysis Providers-Recipe

Each of the MES modules depend on the core Production Module. When any of the MES modules are purchased, the Production Module is included at no additional cost. The Production Module provides the production model functionality that is an object oriented hierarchy of production facilities. It also provides analysis functionality that each specific MES module can extend. The Recipe / Changeover Module provides four analysis providers to analyze differences between recipes, variances and change logs.

The image below shows the impromptu analysis screen where a Recipe C1 6Pk is being compared to its parent Master C. Filters and data points can be selected using the Analysis Selector component, but the Analysis Controller will allow requesting comparisons of recipes without the user interface. The filters and data points are defined through the component properties and is how data for reports is collected.



Value Name	Item Path	Master C_Recipe Value	Recipe C1 6Pk_Recipe Value
Line Speed	Enterprise\Site\Area 1\Line 1	82	82
UCL	Enterprise\Site\Area 1\Line 1\Check...	48.2	48.3
LSL	Enterprise\Site\Area 1\Line 1\Check...	45.0	45.0
LCL	Enterprise\Site\Area 1\Line 1\Check...	45.8	45.8
USL	Enterprise\Site\Area 1\Line 1\Check...	48.0	48.0
Line Speed	Enterprise\Site\Area 1\Line 1\Sealer	0	0
High Temperature Alarm	Enterprise\Site\Area 1\Line 1\Sealer	125.0	125.4
Temperature	Enterprise\Site\Area 1\Line 1\Sealer	115	115
Low Temperature Alarm	Enterprise\Site\Area 1\Line 1\Sealer	110.0	110.0
Leading Delay	Enterprise\Site\Area 1\Line 1\Case...	80	99
Units Per Case	Enterprise\Site\Area 1\Line 1\Case...	24	24
Trailing Delay	Enterprise\Site\Area 1\Line 1\Case...	200	200
Glue Delay	Enterprise\Site\Area 1\Line 1\Case...	2	2
Fill Weight	Enterprise\Site\Area 1\Line 1\Filler	50.75	50.75
Min Fill Level	Enterprise\Site\Area 1\Line 1\Filler	4.0	5.25
Line Speed	Enterprise\Site\Area 1\Line 1\Filler	0	0
Max Fill Level	Enterprise\Site\Area 1\Line 1\Filler	5.5	7.7
Force	Enterprise\Site\Area 1\Line 1\Capper	5.4	9.9
Line Speed	Enterprise\Site\Area 1\Line 1\Capper	0	0
Row Height	Enterprise\Site\Area 1\Line 1\Palleti...	5	5
Pattern Code	Enterprise\Site\Area 1\Line 1\Palleti...	10	10
Spacing Delay	Enterprise\Site\Area 1\Line 1\Labeler	9	9
Line Speed	Enterprise\Site\Area 1\Line 1\Labeler	0	0
Label ID	Enterprise\Site\Area 1\Line 1\Labeler	Y7ST7	Y7ST7

Recipe Analysis

Recipe Analysis Provider

Recipe Analysis Provider

The recipe analysis provider is used to collect recipe values for display or report purposes. By adding multiple Recipe Name filter values, the values will be returned for each recipe name allowing comparing of two or more recipes.

Filters

The recipe analysis provider can accept the following filters:

Category

This is a required filter to specify the type of recipes to return. Only one of the valid options are required:

- Recipe
- Sub Recipe - This includes default values or sub recipes for production items.

Example when using it with the Analysis Controller:

Category=Recipe

Item Path

This is a required filter to specify the production item to include in the results. It is the item path for the desired item path(s). Because analysis is independent of projects, the project name is required in the item path.

Example when using it with the Analysis Controller:

Item Path=RecipeDemo\Enterprise\Site\Area 1\Line 1, Item Path=RecipeDemo\Enterprise\Site\Area 1\Line 2

Children

This is a filter to specify if children of the production item(s) specified in the Item Path filter should be included.Only one of the valid options are required:

- Include
- Exclude (Default)

Example when using it with the Analysis Controller:Children=Include

Format

This is a filter to specify the format of the results.

Only one of the valid options are required:

- None - A row in the results will be created for each recipe included in the Recipe Name filter.

- Recipe Comparison (Default) - This format creates a recipe value column for each recipe included in the Recipe Name filter and consolidates the item path, value name, etc. columns that are common.

Example when using it with the Analysis Controller:

Format=None

Column Naming

Only one of the valid options are required:This is a filter to specify how to name the recipe value columns when the Format filter is set to Recipe Comparison (Default). When this filter is set to Recipe Name Prefix (Default), it is difficult to create a recipe comparison report because the column names change depending on the recipes being compared. Setting this filter to Number Suffix will cause the recipe value column names to always be the same, which simplifies reports.

- Number Suffix
- Recipe Name Prefix (Default)

Example when using it with the Analysis Controller:

Column Naming=Number Suffix

Recipe Name

This is a filter to limit the recipe(s) to include in the results.

Example when using it with the Analysis Controller:

Recipe Name=Recipe A, Recipe Name=Recipe B

Recipe Value Name

This is a filter to limit the recipe value(s) to include in the results.

Example when using it with the Analysis Controller:

Recipe Value Name=Line Speed, Recipe Value Name=Force

Value Types

This is a filter to specify the type of recipe values to include.

Only one of the valid options are required:

- Equal Values - Include only recipe values that match between two or more recipes.

- Not Equal Values - Include only recipe values that do not match between two or more recipes.
- All Values (Default) - Include both values that do not match and do match between two or more recipes.

Example when using it with the Analysis Controller:

Value Types=Not Equal Values

Compare By

The recipe analysis provider does not allow any comparison statements.

Data Points

The recipe analysis provider can accept the following data points:

Assigned By

This is the recipe or production item that assigned the recipe value based on inheritance.

Data Type

This is the data type of the recipe value.

Description

This is the description from the recipe value configuration that was entered in the designer.

Format

This is the numeric format from the associated Ignition tag.

Recipe Name

This is the name of the recipe. If the Format filter is set to Recipe Comparison (Default), then this data point will not be included in the results. This is because columns are added for each recipe being compared. For example when comparing Recipe A to Recipe B, there will be Recipe_A_Recipe_Value and Recipe_B_Recipe_Value columns.

Units

This is the units from the associated Ignition tag.

Recipe Variance Analysis Provider

The recipe variance analysis provider is used to collect recipe variances for display or report purposes. This provider can be used to collect variances in real-time or historically for a date range.

Filters

The recipe variance analysis provider can accept the following filters:

Scope

This is a required filter to specify the scope of what to include in the results:

- Date Range
- Last Active Recipe - This will look for the last recipe selection in the variance log and only include the associated variances in the results. This can be used to monitor active runs or the last run if the recipe has been cancelled.

Example when using it with the Analysis Controller:Scope=Last Active Recipe

Item Path

This is a required filter to specify the production item to include in the results. It is the item path for the desired item path(s). Because analysis is independent of projects, the project name is required in the item path.

Example when using it with the Analysis Controller:

Item Path=RecipeDemo\Enterprise\Site\Area 1\Line 1, Item Path=RecipeDemo\Enterprise\Site\Area 1\Line 2

Children

This is a filter to specify if children of the production item(s) specified in the Item Path filter should be included.

Only one of the valid options are required:

- Include
- Exclude (Default)

Example when using it with the Analysis Controller:

Children=Include

Recipe Name

This is a filter to limit the recipe(s) to include in the results.

Example when using it with the Analysis Controller:

Recipe Name=Recipe A, Recipe Name=Recipe B

Recipe Value Name

This is a filter to limit the recipe value(s) to include in the results.

Example when using it with the Analysis Controller:

Recipe Value Name=Line Speed, Recipe Value Name=Force

Values

This is a filter to specify the type of recipe values to include.

Only one of the valid options are required:

Initial Values - Include only the initial recipe values when the recipe was first selected.

Changed Values - Include only recipe values that changed after the initial values were set.

Both - Include both initial and changed values.

Example when using it with the Analysis Controller:

Values=Both

Compare By

The recipe variance analysis provider does not allow any comparison statements.

Data Points

The recipe variance analysis provider can accept the following data points:

Description

This is the description from the recipe value configuration that was entered in the designer.

From Value

This is the value of the Ignition tag associated with the recipe value before it changed.

Item Path

This is the item path of the production item for the recipe value.

Recipe Name

This is the name of the recipe at the time when the recipe value changed.

Recipe Value

The value that is defined in the recipe.

Time Stamp

The date and time the recipe value changed.

To Value

This is the value of the Ignition tag associated with the recipe value after it changed.

Units

This is the units from the associated Ignition tag.

Value Name

This is the name from the recipe value configuration that was entered in the designer.

Sub Product Code Variance Analysis Provider

The sub product code variance analysis provider is used to collect sub product code variances for display or report purposes. This provider can be used to collect variances in real-time or historically for a date range.

Filters

The recipe variance analysis provider can accept the following filters:

Scope

This is a required filter to specify the scope of what to include in the results:

- Date Range

- Last Active Sub Recipe - This will look for the last sub product code selection in the variance log and only include the associated variances in the results. This can be used to monitor active runs or the last run if the sub recipe has been cancelled.

Example when using it with the Analysis Controller:

Scope=Last Active Sub Recipe

Item Path

This is a required filter to specify the production item to include in the results. It is the item path for the desired item path(s). Because analysis is independent of projects, the project name is required in the item path.

Example when using it with the Analysis Controller:

Item Path=RecipeDemo\Enterprise\Site\Area 1\Line 1, Item Path=RecipeDemo\Enterprise\Site\Area 1\Line 2

Children

This is a filter to specify if children of the production item(s) specified in the Item Path filter should be included.

Only one of the valid options are required:

- Include
- Exclude (Default)

Example when using it with the Analysis Controller:

Children=Include

Recipe Value Name

This is a filter to limit the recipe value(s) to include in the results.

Example when using it with the Analysis Controller:

Recipe Value Name=Line Speed, Recipe Value Name=Force

Sub Recipe Name

This is a filter to limit the sub recipe(s) to include in the results.

Example when using it with the Analysis Controller:

Recipe Name=1A, Recipe Name=1B

Values

This is a filter to specify the type of recipe values to include.

Only one of the valid options are required:Initial Values - Include only the initial recipe values when the recipe was first selected.Changed Values - Include only recipe values that changed after the initial values were set.Both - Include both initial and changed values.

Example when using it with the Analysis Controller:

Values=Both

The sub product code variance analysis provider does not allow any comparison statements.

Data Points

The sub product code variance analysis provider can accept the following data points:

Description

This is the description from the recipe value configuration that was entered in the designer.

From Value

This is the value of the Ignition tag associated with the recipe value before it changed.

Item Path

This is the item path of the production item for the recipe value.

Sub Recipe Name

This is the name of the sub recipe at the time when the recipe value changed.

Recipe Value

The value that is defined in the recipe.

Recipe Value Name

This is the name from the recipe value configuration that was entered in the designer.

Time Stamp

The date and time the recipe value changed.

To Value

This is the value of the Ignition tag associated with the recipe value after it changed.

Units

This is the units from the associated Ignition tag.

Recipe Change Log Analysis Provider

The recipe change log analysis provider is used to collect the change log entries for display or report purposes. All details of changes made to recipes including adding new recipes, adding production items to recipes and much more can be returned using the recipe change log analysis provider.

Filters

The recipe change log analysis provider can accept the following filters:

Category

This is a required filter to specify the type of recipes to return. One or more of the valid options are required:

Recipe - This includes all recipe changes excluding value changes.

Recipe Value - This includes only recipe value changes.

Sub Recipe - This includes default value or sub recipe changes excluding value changes.

Sub Recipe Value - This includes default value or sub recipe value changes.

Example when using it with the Analysis Controller:

Category=Recipe, Category=Recipe Value

Item Path

This is a required filter to specify the production item to include in the results. It is the item path for the desired item path(s).

Example when using it with the Analysis Controller:

Item Path=Enterprise\Site\Area 1\Line 1, Item Path=Enterprise\Site\Area 1\Line 2

Children

This is a filter to specify if children of the production item(s) specified in the Item Path filter should be included.

Only one of the valid options are required:

Include

Exclude (Default)

Example when using it with the Analysis Controller:

Children=Include

Recipe Name

This is a filter to limit the recipe(s) to include in the results.

Example when using it with the Analysis Controller:

Recipe Name=Recipe A, Recipe Name=Recipe B

Recipe Value Name

This is a filter to limit the recipe value(s) to include in the results.

Example when using it with the Analysis Controller:

Recipe Value Name=Line Speed, Recipe Value Name=Force

Sub Recipe Name

This is a filter to limit the sub recipe(s) to include in the results.

Example when using it with the Analysis Controller:

Recipe Name=1A, Recipe Name=1B

Compare By

The recipe analysis provider does not allow any comparison statements.

Data Points

The recipe analysis provider can accept the following data points:

Change Type

This is a description of the type of change. For example: it can be Recipe value changed or Recipe value reverted.

Changed By

This is the person that made the change.

Description

This is the description from the recipe value configuration that was entered in the designer.

From Value

This is the value before the change.

Info

This is additional information that further describes the change.

Item Path

This is the item path of the production item that the change was made for.

Note

This is the note that was entered by the user at the time of the change.

Recipe Name

This is the name of the recipe the change was made for.

Sub Product Code

This is the sub product code the change was made for.

Time Stamp

The date and time of the changed.

To Value

This is the value after the change.

Value Name

This is the name from the recipe value configuration that was entered in the designer.

Units

This is the units from the associated Ignition tag.

8.4.9 Production OPC Values-Recipe

This references details the production OPC values that the Recipe / Changeover Module provides. For each property, the Ignition data type is listed and if it is read only. The Ignition data types correspond to the data types that are available for SQLTags.

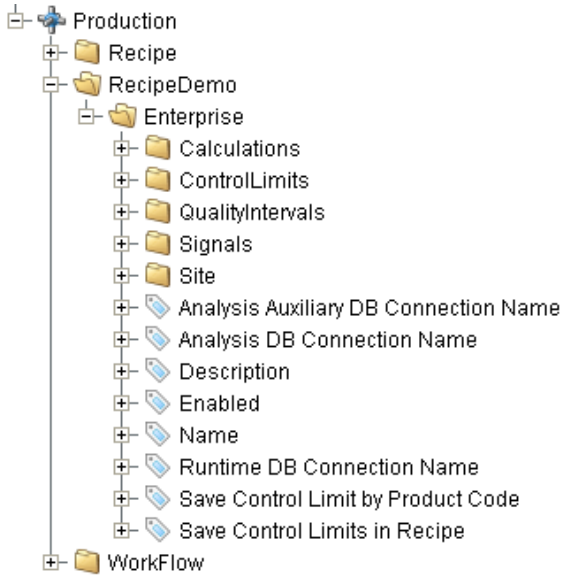
Within this reference, the **Read Only** means that the OPC value cannot be written to through the OPC Production Server. It can only be set in the designer or it is a calculated value. Trying to write to a read only property will result in an error message being shown.

Depending on the MES modules that are installed into the Ignition server, more or less production OPC values will appear when browsing. For example, if only the Recipe / Changeover Module is installed, then only production OPC values that the core MES or Recipe / Changeover Module provide will appear.

Enterprise-Recipe

Description

The enterprise folder contains some properties associated with the enterprise and a folder for each production Site within it. The name is the same as the enterprise name that is configured in the designer. The image below represents the **Enterprise** of the RecipeDemo project.



Recipe OPC Enterprise

Enterprise

Child Folders

Site	One folder will exist for each Site that has been configured in the Ignition Designer. The folder can be opened to view all values within the site.

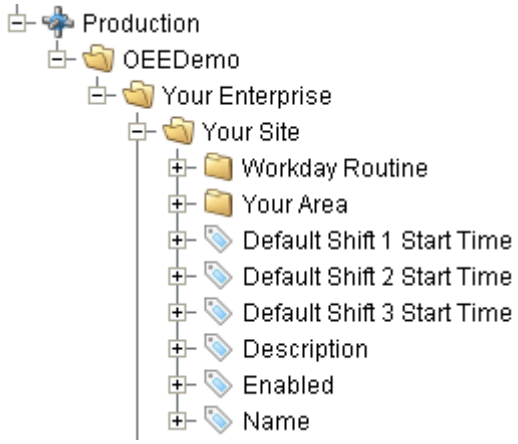
Properties

Analysis Auxiliary DB Connection Name	The name of the auxiliary (mirror) analysis database connection. Can be blank if no auxiliary DB connection is configured.	String Read Only
Analysis DB Connection Name	The name of the analysis database connection.	String Read Only
Description	Optionally, this property can be set to a description for the enterprise. It is not used by the MES modules other than for reference.	String
Enabled	This reflects the enterprise Enabled property in the Designer. If the enterprise Enabled is set to true, then the MES production model will perform calculations for the enterprise and all sites, areas, lines, cells, cell groups and location within it. If this property is set to false, then none of the sites, areas, lines, cells, cell groups and locations will have calculations performed.	Boolean
Name	This reflects the name of the enterprise that is set in the designer.	String Read Only
Runtime DB Connection Name	The name of the runtime database connection.	String Read Only

Site-Recipe

Description

The site folder contains some properties associated with the production site and a folder for each production area within it. The name is the same as the site name that is configured in the designer. The image below represents the **Your Site** of the OEEDemo project.



OPC Site Node

Site

Child Folders

Area	One folder will exist for each area that has been configured in the Ignition Designer. The folder can be opened to view all values within the area.
RecipeValue	Any recipe values that are configured for the production site will appear in this folder.

Properties

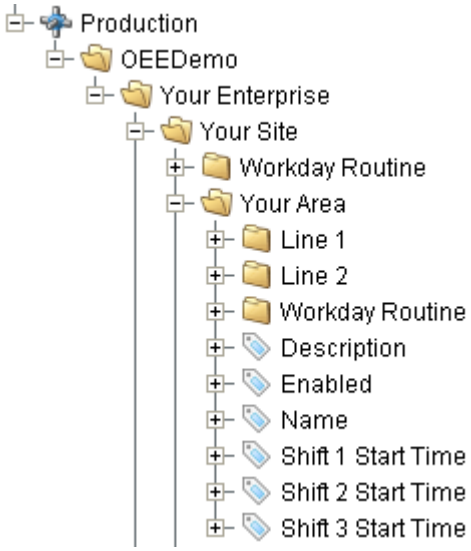
Description	Optionally, this property can be set to a description for the site. It is not used by the OEE Downtime and Scheduling Module other than for reference.	String
Enabled		Boolean

	This reflects the site Enabled property in the Designer. If the site Enabled is set to true, then the OEE Downtime and Scheduling module will perform calculations for the site and all areas, lines and cells within it. If this property is set to false, then none of the areas, lines or cells will have calculations performed.	
Name	This reflects the name of the site that is set in the designer.	String Read Only

Area-Recipe

Description

The area folder contains some properties associated with the production area and a folder for each production line within it. The name is the same as the area name that is configured in the designer. The image below represents the **Your Area** of the OEEDemo project.



OPC Area Node

Area

Child Folders

Line	One folder will exist for each Line that has been configured in the Ignition Designer. The folder can be opened to view all values within the line.

RecipeValue	Any recipe values that are configured for the production area will appear in this folder.

Properties

Description	Optionally, this property can be set to a description for the area. It is not used by the OEE Downtime and Scheduling Module other than for reference.	String
Enabled	This reflects the site Enabled property in the Designer. If the area Enabled is set to true, then the OEE Downtime and Scheduling module will perform calculations for the area and all lines and cell within it. If this property is set to false, then none of the lines or cells will have calculations performed.	Boolean
Name	This reflects the name of the area that is set in the designer.	String Read Only

Line-Recipe

Description

The line folder contains some properties associated with the production line and a folder for each production cell within it. The name is the same as the line name that is configured in the designer. The image below represents the **Line 1** of the OEEDemo project.

OPC Line Node

Line

Child Folders

RecipeValue	Any recipe values that are configured for the production line will appear in this folder.
Cell	One folder will exist for each Cell that has been configured in the Ignition Designer. The folder can be opened to view all values within the cell.

Properties

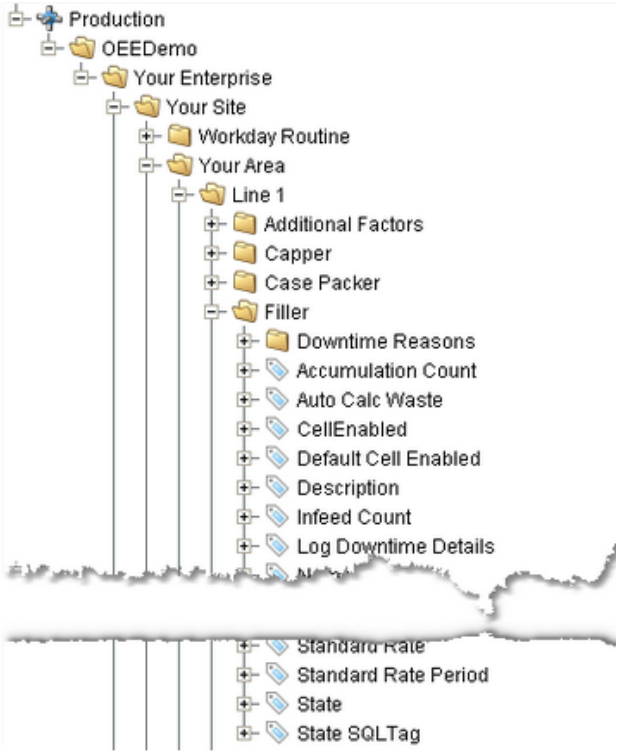
ActiveRecipeName	If a recipe is active for this production line, then this is the name of the recipe. If a recipe is not active, then this is blank.	String Read Only
RecipeLoading	True if a recipe is currently being loaded for the production line.	Boolean

		Read Only
RecipeActive	Indicates if a recipe is currently active.	Boolean Read Only
RecipeScale	Set this to the amount to scale a recipe prior to selecting a recipe for the production line.	Double
RecipeTrackingUUID	This is a unique value used for tracking initial recipe values and variances while a recipe is selected. It can be used when looking up data directly from the database.	String Read Only
RecipeVariancesExists	If true, then Ignition tags associated with at least one recipe value for this production item have changed.	Boolean Read Only
RecipeWriteError	If true, then at least one recipe value did not write to the associated Ignition tags when the recipe was first selected.	Boolean Read Only
ValueMonitorEnabled	If true, recipe values are being monitored and recipe value variances will be logged.	Boolean Read Only
EnableRecipe	Set to true to allow recipes to be selected for this production item. This is useful when selecting a recipe for a production line and preventing selecting the same recipe for selected child production items.	Boolean

Cell Group-Recipe

Description

The cell folder contains some properties associated with the production cell. The name is the same as the cell name that is configured in the designer. The image below represents the Filler of the OEEDemo project.



OPC Cell Node

Cell

Child Folders

Cell	One folder will exist for each Cell that has been configured in the Ignition Designer. The folder can be opened to view all values within the cell.
RecipeValue	Any recipe values that are configured for the production cell group will appear in this folder.

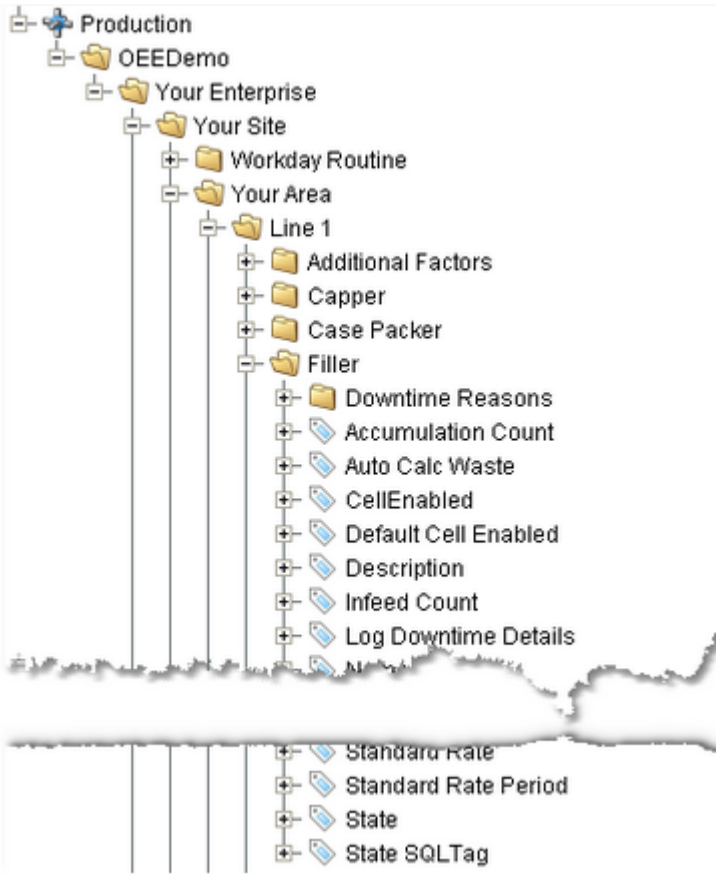
Properties

ActiveRecipeName	If a recipe is active for this production cell, then this is the name of the recipe. If a recipe is not active, then this is blank.	String Read Only
RecipeLoading	True if a recipe is currently being loaded for the production cell group.	Boolean Read Only
RecipeActive	Indicates if a recipe is currently active.	Boolean Read Only
RecipeScale	Set this to the amount to scale a recipe prior to selecting a recipe for the production cell group.	Double
RecipeTrackingUUID	This is a unique value used for tracking initial recipe values and variances while a recipe is selected. It can be used when looking up data directly from the database.	String Read Only
RecipeVariancesExists	If true, then Ignition tags associated with at least one recipe value for this production item have changed.	Boolean Read Only
RecipeWriteError	If true, then at least one recipe value did not write to the associated Ignition tags when the recipe was first selected.	Boolean Read Only
ValueMonitorEnabled	If true, recipe values are being monitored and recipe value variances will be logged.	Boolean Read Only
EnableRecipe	Set to true to allow recipes to be selected for this production item. This is useful when selecting a recipe for a production line and preventing selecting the same recipe for selected child production items.	Boolean

Cell-Recipe

Description

The cell folder contains some properties associated with the production cell. The name is the same as the cell name that is configured in the designer. The image below represents the Filler of the OEEDemo project.



OPC Cell Node

Cell

Child Folders

RecipeValue	Any recipe values that are configured for the production cell will appear in this folder.

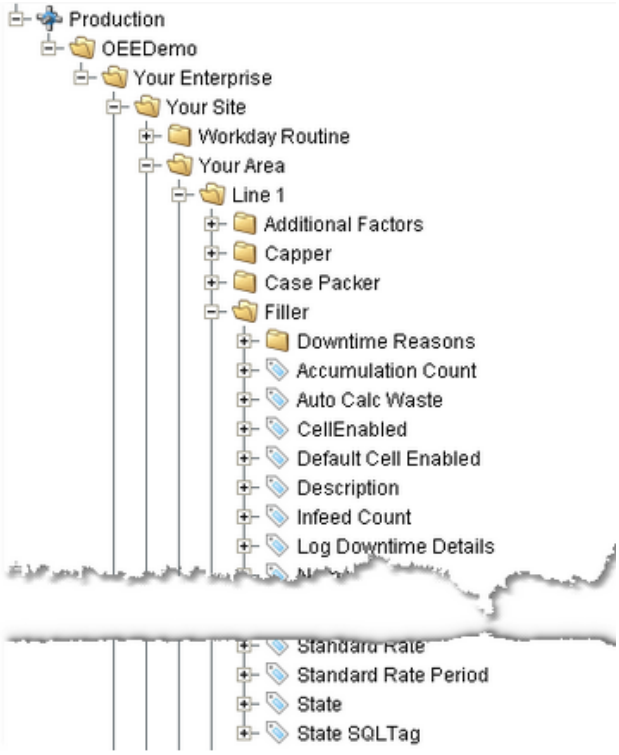
Properties

ActiveRecipeName	If a recipe is active for this production cell, then this is the name of the recipe. If a recipe is not active, then this is blank.	String Read Only
RecipeLoading	True if a recipe is currently being loaded for the production cell.	Boolean Read Only
RecipeActive	Indicates if a recipe is currently active.	Boolean Read Only
RecipeScale	Set this to the amount to scale a recipe prior to selecting a recipe for the production cell.	Double
RecipeTrackingUUID	This is a unique value used for tracking initial recipe values and variances while a recipe is selected. It can be used when looking up data directly from the database.	String Read Only
RecipeVariancesExists	If true, then Ignition tags associated with at least one recipe value for this production item have changed.	Boolean Read Only
RecipeWriteError	If true, then at least one recipe value did not write to the associated Ignition tags when the recipe was first selected.	Boolean Read Only
ValueMonitorEnabled	If true, recipe values are being monitored and recipe value variances will be logged.	Boolean Read Only
EnableRecipe	Set to true to allow recipes to be selected for this production item. This is useful when selecting a recipe for a production line and preventing selecting the same recipe for selected child production items.	Boolean

Location-Recipe

Description

The cell folder contains some properties associated with the production cell. The name is the same as the cell name that is configured in the designer. The image below represents the Filler of the OEEDemo project.



OPC Cell Node

Cell

Child Folders

RecipeValue	Any recipe values that are configured for the production location will appear in this folder.

Properties

ActiveRecipeName	If a recipe is active for this production cell, then this is the name of the recipe. If a recipe is not active, then this is blank.	String Read Only

RecipeLoading	True if a recipe is currently being loaded for the production location.	Boolean Read Only
RecipeActive	Indicates if a recipe is currently active.	Boolean Read Only
RecipeScale	Set this to the amount to scale a recipe prior to selecting a recipe for the production location.	Double
RecipeTrackingUUID	This is a unique value used for tracking initial recipe values and variances while a recipe is selected. It can be used when looking up data directly from the database.	String Read Only
RecipeVariancesExists	If true, then Ignition tags associated with at least one recipe value for this production item have changed.	Boolean Read Only
RecipeWriteError	If true, then at least one recipe value did not write to the associated Ignition tags when the recipe was first selected.	Boolean Read Only
ValueMonitorEnabled	If true, recipe values are being monitored and recipe value variances will be logged.	Boolean Read Only
EnableRecipe	Set to true to allow recipes to be selected for this production item. This is useful when selecting a recipe for a production line and preventing selecting the same recipe for selected child production items.	Boolean

8.4.10 Copy of Grouping recipes

Overview

The following methods can be used for grouping recipes by product type, by master recipes, by production item respectively.

Applies To and Version Info

This feature applies to Recipe module and is available in all versions.

Scenario

Line 1 has Nutella_Recipe 1, Nutella_Recipe 2, Nutella_Recipe 3 (***Descendants under master Nutella***)

Line 2 has estathe_Recipe 10, estathe_Recipe 11, estathe_Recipe 12 (***Descendants under master estathe***)

Method 1: Grouping by product type

The getItemRecipeList script function has a recipe filter parameter that can be used to limit the recipe names that are returned. If the recipe names start with the product, then the recipes can be limited by Nutella, estathe, etc.

Calling getItemRecipeList('Enterprise\Site\Area\Line 1', 'Nutella_*') script function will only return Nutella_Recipe 1, Nutella_Recipe 2, Nutella_Recipe 3. And calling getItemRecipeList ('Enterprise\Site\Area\Line 1', 'estathe_*') script function will only return estathe_Recipe 10, estathe_Recipe 11, estathe_Recipe 12.

```
itemPath = 'Enterprise\Site\Area\Line 1'

#Request the available recipes with specified prefix.
list = system.recipe.getItemRecipeList(itemPath, "Nutella_*")
```

Method 2: Grouping by master recipe

Setup master recipes for Nutella, estathe, etc. and then add the actual recipes as descendants under the appropriate master recipe. The getItemRecipeList script function recipe filter parameter will limit recipe names that are returned to those that belong to the master. For example: If you have the following recipes defined Nutella (Master) -> Recipe 1 (Descendant under master Nutella) -> Recipe 2 (Descendant under master Nutella) -> Recipe 3 (Descendant under master Nutella) estathe (Master) -> Recipe 10 (Descendant under master estathe) -> Recipe 11 (Descendant under master estathe) -> Recipe 12 (Descendant under master estathe) Then, calling getItemRecipeList('Enterprise\Site\Area\Line 1', 'Nutella*') script function will only return Recipe 1, Recipe 2, Recipe 3. And calling getItemRecipeList ('Enterprise\Site\Area\Line 1', 'estathe*') script function will only return Recipe 10, Recipe 11, Recipe 12.

```
itemPath = 'Enterprise\Site\Area\Line 1'

#Request the available recipes for the selected Master recipe
list = system.recipe.getItemRecipeList(itemPath, "estathe*")
```

Method 3: Grouping by production item

Use separate lines and cells for each product (even though physically there may not be two lines). For example: Recipe 1 - Line 1 Recipe 2 - Line 1 Recipe 3 - Line 1 Recipe 10 - Line 2 Recipe 11 - Line 2 Recipe 12 - Line 2 Then calling getItemRecipeList('Enterprise\Site\Area\Line 1', '') script function will only return Recipe 1, Recipe 2, Recipe 3. Calling getItemRecipeList ('Enterprise\Site\Area\Line 2', '') script function will only return Recipe 10, Recipe 11, Recipe 12 .

```
itemPath = 'Enterprise\Site\Area\Line 1'

#Request the available recipes for Line 1
list = system.recipe.getItemRecipeList(itemPath, '')
```

Method 1 and 2 don't require any additional tracking points and it is more straight forward and less problematic. Use method 1 if you have a prefix on the recipe name.

References

system.recipe.getItemRecipeList

Master Recipes

Production Item

Created By: Jason Coope **Created Date:** May 04, 2017 09:07**Last Modified By:** Jason Coope **Last Modifed Date:** May 04, 2017 09:07

8.4.11 Copy of Create recipe in scripting

Overview

This sample script shows how to create a recipe and assign values to the recipe items.

Applies To and Version Info

This feature applies to Recipe module and is available in all versions.

Scripting

If this script function is placed in the shared script library under a package called "operations" and a script library called "RecipeManagement" it would be executed with the call:
"shared.operations.RecipeManagement.createRecipe()"

Script function can have parameters so values such as recipe name and item path may be passed in with the call.

```
 def createRecipe():
    from org.apache.log4j import Logger
    log = Logger.getLogger('createRecipeScriptLogger')

    # create the recipe
    recipeName = 'Manual Recipe 7'
    parentRecipeName = ''
    note = 'Manual created recipe'
    system.recipe.createRecipe(recipeName, parentRecipeName, note)

    # assign production item to recipe
    itemPath = 'Your Enterprise\Site 1\Packaging\Line 1'
    system.recipe.addItemToRecipe(recipeName, itemPath, note)

    # display current recipe values
    category = '1' # indicates recipe values were created viw the
recipe module
    recipeVals = system.recipe.getRecipeValues(itemPath,
recipeName, category)
    log.info('Original recipe values')
    log.info('_____
_')
    for ndx in range(recipeVals.size()):
        recipeItem = recipeVals.get(ndx) # get the recipe value
item object #recipeItem = recipeVals.get(ndx).getRecipeValue() #
if used in gateway scope
        itemName = recipeItem.getName()
        itemValue = str(recipeItem.getValue())
        #itemSort = recipeItem.getSortOrder()
        #recipeItem.getMinValue()
```

```
        #recipeItem.getMaxValue()
        #recipeItem.getAssignedBy()
        #recipeItem.getSortOrder()
        #recipeItem.hasDescription()
        #recipeItem.getDescription()
        #recipeItem.getFormat()
        #recipeItem.getUnits()
        log.info('%s=%s' %(itemName, itemValue))

    # assign values to the recipes items
    valueName = 'IntRecipeTag'
    value = '346' # always assign as a string, the module will
convert to the proper type
    note = 'value changed'
    system.recipe.setPathRecipeValue(itemPath, recipeName,
valueName, value, note)
    valueName = 'StringRecipeTag'
    value = 'Updated string value'
    note = 'value changed'
    system.recipe.setPathRecipeValue(itemPath, recipeName,
valueName, value, note)

    # display new values
    recipeVals = system.recipe.getRecipeValues(itemPath,
recipeName, category)
    log.info('Modified recipe values')
    log.info('_____________________________________________________
_')
    for ndx in range(recipeVals.size()):
        recipeItem = recipeVals.get(ndx)
        itemName = recipeItem.getName()
        itemValue = str(recipeItem.getValue())
        log.info('%s=%s' %(itemName, itemValue))
    return
```

References

system.recipe.createRecipe

system.util.getLogger

system.recipe.addItemToRecipe

system.recipe.getRecipeValues

system.recipe.setPathRecipeValue

Keywords

Set recipe values

Created By: Jason Coope **Created Date:** May 04, 2017 09:09**Last Modified By:** Jason Coope
Last Modifed Date: May 04, 2017 09:09

8.4.12 Copy of Recipe Value Security Settings

Overview

This kb article illustrates how to set security role settings for a recipe value and how to revert the inherit property of the recipe value security settings through scripting.

Applies To and Version Info

This feature applies to Recipe module and is available in all versions.

Security role

The list of security roles and the details of individual security role can be retrieved through scripting.

Set inherit

setInherit() property can be used to change security settings of a recipe item.

Scripting

```
linePath = event.source.parent.getComponent('MES Object Selector')
.equipmentItemPath

# Get a list of all the recipe entries under this path
recipeVals = system.recipe.getDefaultValues(linePath, "1", "")

# Cycle through the list
for ndx in range(recipeVals.size()):

    # Get the recipe item at this point
    recipeItem = recipeVals.get(ndx)

    # Get the name of the item
    recipeItemName = recipeItem.getName()
    print recipeItemName

    # Get the security object for this item
    secInfo = system.recipe.getRecipeValueSecurity(linePath,
recipeItemName, False)
```

```
    #Cycle through and print the setting for each role
    #for ndx in range(secInfo.getSecurityRoleCount()):
    #   recSec = secInfo.getSecurityRole(ndx)
    #   print recSec.getSecurityRole()
    #   print recSec.isAllowEdit()

    # Get the security info for a specific role
    secRole = secInfo.getSecurityRole('Supervisor')

    # Allow the role to edit
    secRole.setAllowEdit(True)

    #This must be set otherwise it will inherit from the parent
    secInfo.setInherit(False)
    print secRole.getMinValue()
    print secRole.getMaxValue()

    # if you know the datatype or you know the item by name you
can set the min and max for the role
    #if recipeItemName == 'FanSpeed':
    #   secRole.setMinValue(32.5)
    #   secRole.setMaxValue(212.0)
    #print

    #Update the security settings
    system.recipe.updateRecipeValueSecurity(secInfo)
```

References

Recipe Value Security Role

Recipe Value Security Info

system.recipe.getDefaultValues

system.recipe.getRecipeValueSecurity

system.recipe.updateRecipeValueSecurity

Keywords

Recipe setInherit property

Created By: Jason Coope **Created Date:** May 04, 2017 09:09**Last Modified By:** Jason Coope **Last Modifed Date:** May 04, 2017 09:09

8.5 Instrument Interface

New to Instrument Interface?

Download and testdrive this most powerful MES solution available anywhere!

Download and Install Module

Instrument Interface Module in a nutshell

Click on the topic you would like to learn more about ...

- Components
- Scripting
- Objects

Product Data Sheet

To see the Product Data Sheet,

click on **Instrument Interfac e Module**

Click here to see Knowledge base articles of Instrument Interface.

Info

Sepasoft Instrument Interface module uses system.instrument functions for scripting.

Read this section about licensing...

Licensing and Activation

8.5.1 Instrument Interface Overview

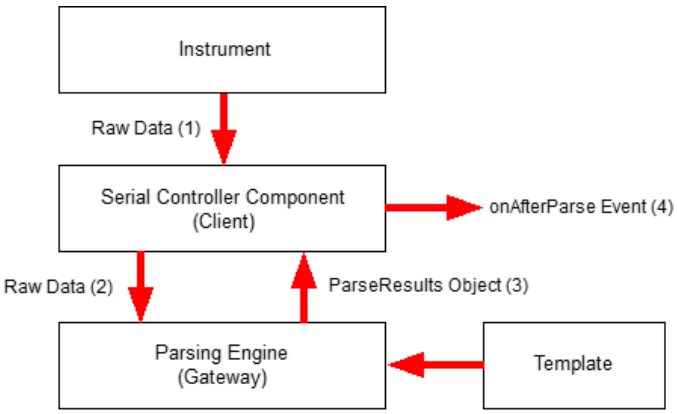
The Instrument Interface module easily interfaces with different instruments, providing you with data collection from instruments such as gauges, analyzers, and barcode scanners. You can capture raw textual data from instrument type devices via serial, text files, OPC devices and more, enabling you to parse out the meaningful values that can be used in your Ignition application, or save the data directly to a database, or pass it along to other systems. This module provides a seamless solution to eliminating hundreds of lines of code and extracting only the meaningful values from devices connected to your SCADA system.

Module Features:

- Serial Communications
- File Monitoring
- Parsing
- Centralized Instrument Management

The Instrument Interface module is used to define communication settings and data parsing templates to an instrument. These settings and parsing templates are then used to read data from a instrument and parse the raw data to extract desired values. The data from an instrument can come from a file, serial communication port, TCP or UDP connection, OPC device such as a PLC, external data or web service.

The image show the typical flow of data when reading instrument values through a serial communications port. Note that the Client Serial Support Module is required to read serial data on a client computer.

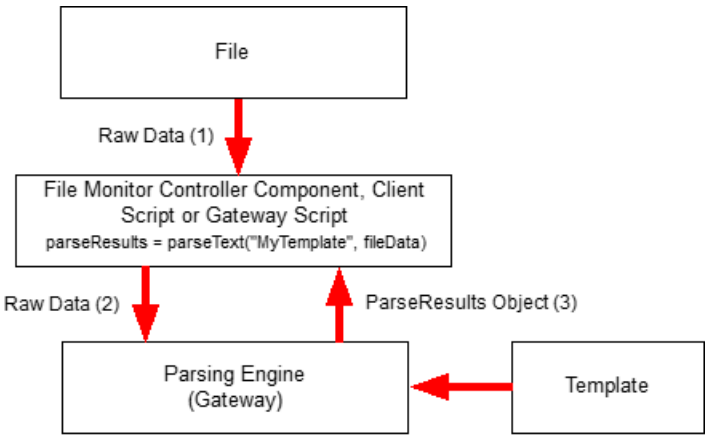


SerialParseFlow

Typical Serial Communications Flow

The Instrument Interface Module includes a component to make configuring and control of serial port communications easier than using the script only support of the Client Serial Support Module. If reading data from a serial communication port on the Ignition server is needed, then the Serial Server Support Module is needed.

Some Instruments write their results to a disk file. The image shows the typical flow when reading data from a file and using the File Monitor component or parsing script functions that are available on both the client and the gateway to parse the raw data in the file.



FileParseFlow

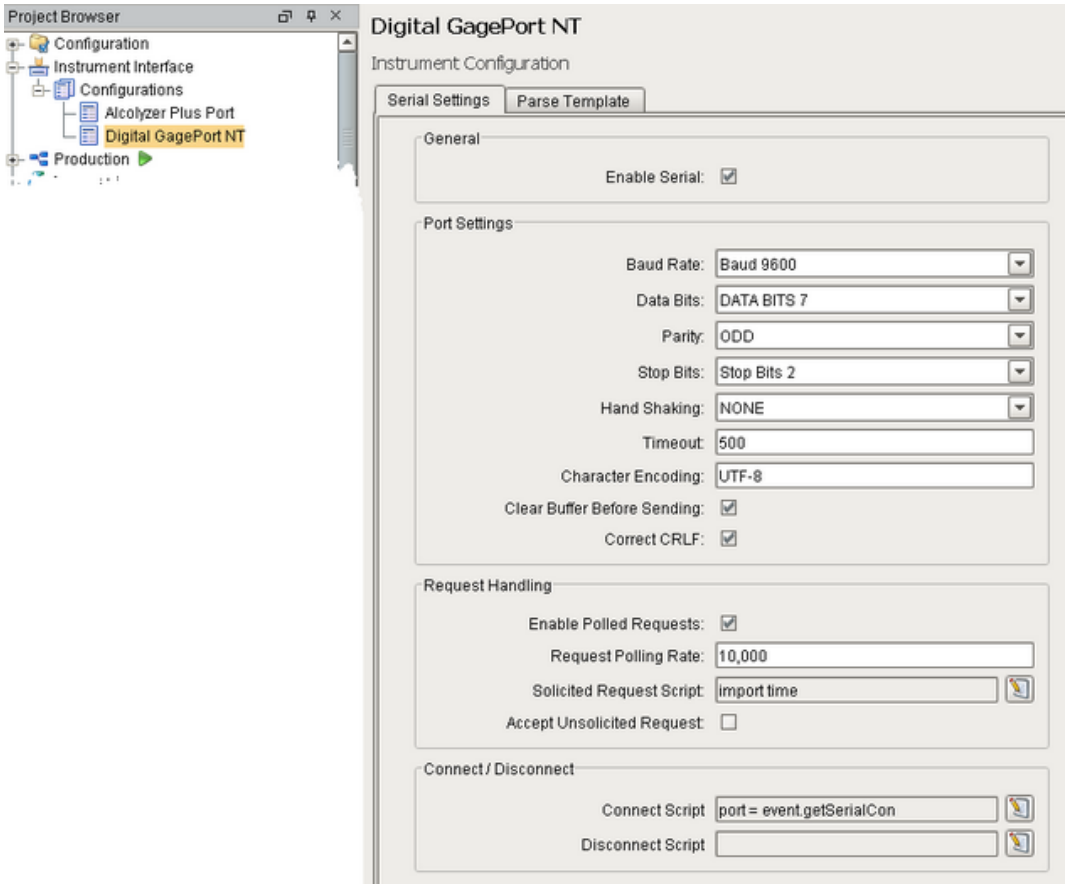
Typical File Flow

8.5.2 Instrument Interface Configurations

Configure the instrument interface by defining communication settings and data parsing templates to an instrument. The settings and parsing templates are then used to read data from an instrument and parse the raw data to extract desired values. The data from an instrument can come from a file, serial communication port, TCP or UDP connection, OPC device such as a PLC, external data, or web service.

Serial Settings

This page configures the serial port communications settings of this Instrument Interface.



Serial Settings Configuration

The Instrument Interface Module is perfect for serial devices – such as analyzers, measurement gauges, barcode readers and many more – where values are sent when the operator presses the send button or a request is sent to the device to read the values. The serial support built into the Instrument Interface Module includes polling the device for new data, receiving unsolicited data from the device, or requesting the new data from the device based on an event in Ignition. It gracefully handles timeouts and other communication issues that are common with serial communications and also includes the flexibility of controlling every byte sent or received using script.

General

Enable Serial	If checked, these port settings will be applied to the Serial Controller component when this Instrument Interface is assigned to its Instrument Interface Name property.

Port Settings

Baud Rate	Serial Communication baud rate. Select from the following: • Baud 110 • Baud 150 • Baud 300 • Baud 600 • Baud 1200 • Baud 2400 • Baud 4800 • Baud 9600 • Baud 19200 • Baud 38400 • Baud 57600 • Baud 115200 • Baud 230400 • Baud 460800 • Baud 921600
Data Bits	Serial communication data bits. Select from the following: • DATA BITS 5 • DATA BITS 6 • DATA BITS 7 • DATA BITS 8
Parity	Serial communication parity. Select from the following: • NONE • EVEN • ODD • MARK • SPACE

Stop Bits	Serial communication number of stop bits. Select from the following: • Stop Bits 1 • Stop Bits 2
Hand Shaking	Serial communication flow control methods. Select from the following: • NONE • CTS DTR • CTS RTS • DSR DTR • XON XOFF
Timeout	The default number of milliseconds to wait while reading data.
Character Encoding	Character encoding of the data.
Clear Buffer Before Sending	If checked, clears the receive buffer before sending data.
Correct CRLF	If checked, corrects any combination of end of line characters to carriage return (CR) and line feed (LF).

Request Handling

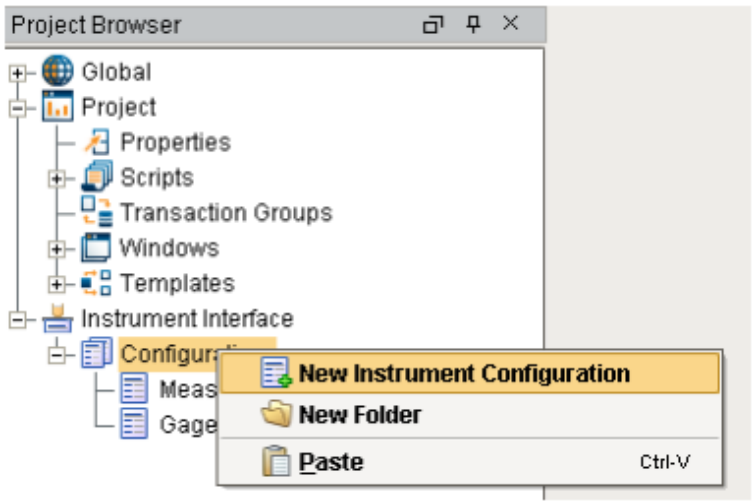
Enable Polled Requests	If checked, the port will be polled at the requested rate.
Request Polling Rate	The rate in milliseconds to poll the port.

Solicited Request Script	The script to run for each polled request.
	When writing scripts you can use the "event" object to reference methods in the Serial Controller component that this Instrument Interface is assigned.
	Code Snippet
	`import time`
	`port = event.getSerialController()`
	`port.clearBuffer()`
	`port.writeString("Ar")`
	`time.sleep(0.5)`
	`port.writeString("As")`
	`time.sleep(0.5)`
	`event.setReceivedData(port.readString())`

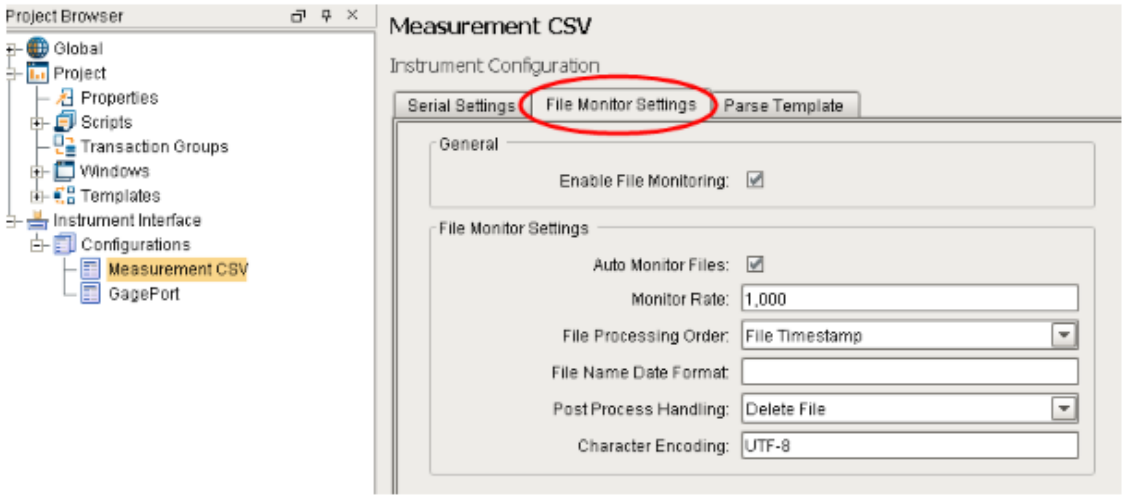
Accept Unsolicited Request	If checked, the port will can accept requests without being solicited.

File Monitor Settings

This page configures the file monitor settings of this Instrument Interface. It provides a configuration area by instrument type that use file method of handing off data. Since we don't have a serial device, let's configure a simple file parser using the Instrument Interface module. Open up the quality project in the Ignition designer. Right-click on the **Configurations** menu item under **Instrument Interface** in the **Project Browser** to add a new file monitoring configuration.



There should already be two configurations in the quality project: Measurement CSV and GagePort. Click on the **Measurement CSV** configuration. Since we are configuring file monitoring, click on the **File Monitor Settings** tab.



Here you can enable file monitoring and specify the settings. Use the settings above to automatically monitor files in a directory every second. If files exists we will process them by their timestamp and delete them once we are done parsing. The next step is to configure a parse template so Ignition knows how to parse the files. You can check out the GagePort for information on serial parsing.

Some instruments only support passing data through the use of a file; the Instrument Interface Module makes the process easy. You can also read values from external software programs that only support passing data through the use of files. The format of the data can vary from a reports format, CSV (comma separated values) or even a mixture of the two.

General

Enable File Monitoring	If checked, these file monitoring settings will be applied to File Monitor component when this Instrument Interface is assigned to its Instrument Interface Name property.

File Monitor Settings

Auto Monitor Files	If true automatically detects and processes file(s) contained with the File Path property of the File Monitor component. If false, the read() of the component must be called to process file(s).
Monitoring Rate	The milliseconds between each check for new files. Any files that are found during a check will be processed. Processing of file will not overlap. If the time it takes to process the files exceeds the value of this property, then the next check will be at the next interval.
File Processing Order	This property defines the priority to process multiple file. It is inapplicable when a single file is selected in the File Path property. If Alpha Numeric is selected, the files are processed in alphabetical order. If Date is selected, the file names are converted to date values using the pattern defined in the File Name Date Format property and then processed in chronological order. If File Timestamp is selected, the files are processed in chronological order of the file modified date. Select from the following: Alpha Numeric = 0 Date = 1 File Timestamp = 2
File Name Date Format	This property is only applicable if the File Processing Priority property is set to Date. This property defines the parsing pattern to use when converting the file name to a date value when determining the processing order of the files. The patterns can contain both date and time format designators. See the File Name Date Format property description of the File Monitor component for more details.
After Processing Handling	This setting defines how files are handled after processing them. Select from the following: Delete File = 0 Move File = 1
Character Encoding	Character encoding of the data.

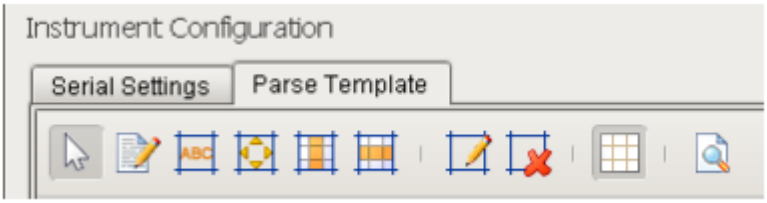
Parse Template

The Parse template allows a visual way of defining the individual data points to extract from the raw text returned from the instrument interface. The text represents what is returned from an instrument and is displayed in a fixed character width. Multiple parsing boxes can be added to define areas to extract meaningful values from.

The Parse Template tools are explained in detail below.

Parse Template Tools

The Parse Template configuration screen contains a toolbar palette with tools that allow interaction with the current parse template.

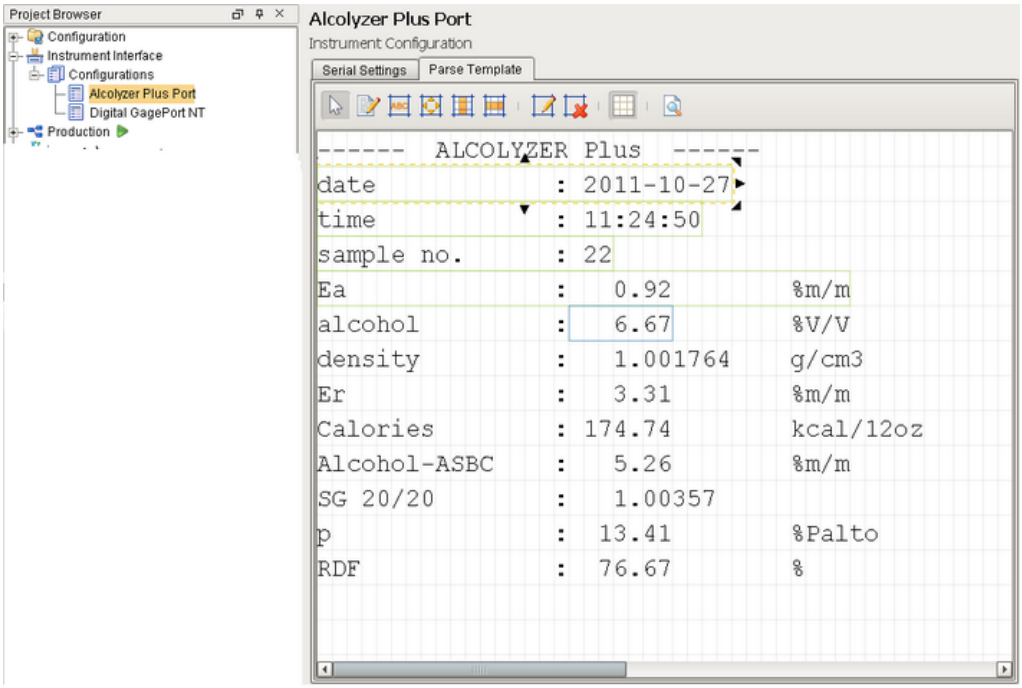


Parsing Box Selector

Allows the user to select any existing parsing boxes in the template. The selected parsing box will be displayed with sizing arrows on all corners.

Edit Content Text

Allows editing of the actual template text that the parsing operations will be applied to. New templates are blank and the user will need to add text representing the output received from the instrument here so that parsing boxes can be applied. This can also be populated by using the "Send to Template" menu option of the Serial Controller and File Monitor Components. Refer to the the documentation for these components for more information.

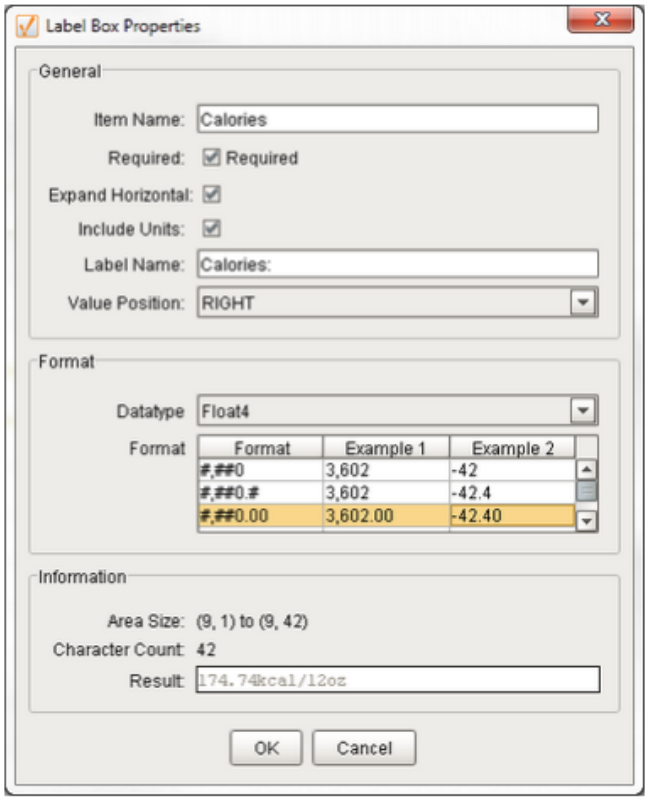


ParseTemplate

Parse Template Configuration Screen

Find Label Parsing Box

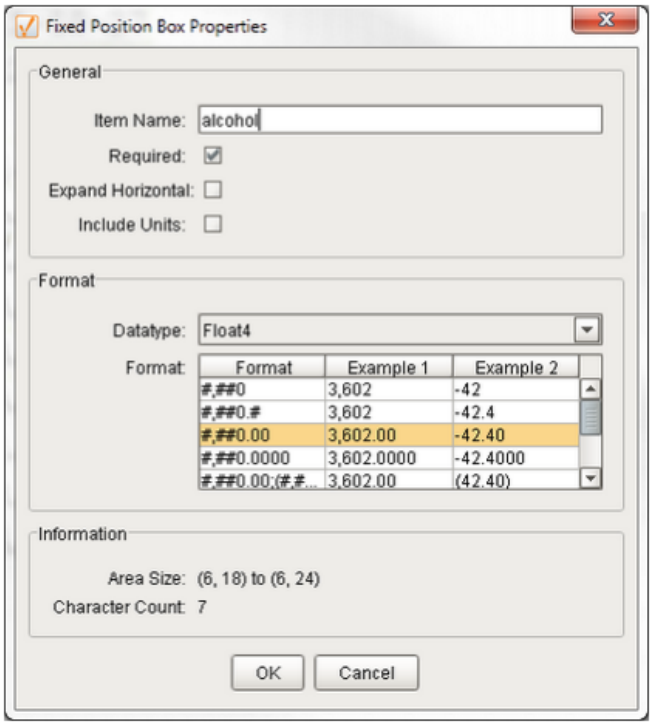
The parsed data will be linked to a label on the template so that the position of the label/value pair can appear at any location.



Label Parsing Box

Fixed Position Parsing Box

The parsed data will be read from the template at the exact position the box is placed.



Fixed Position Parsing Box

CSV Column Parsing Box

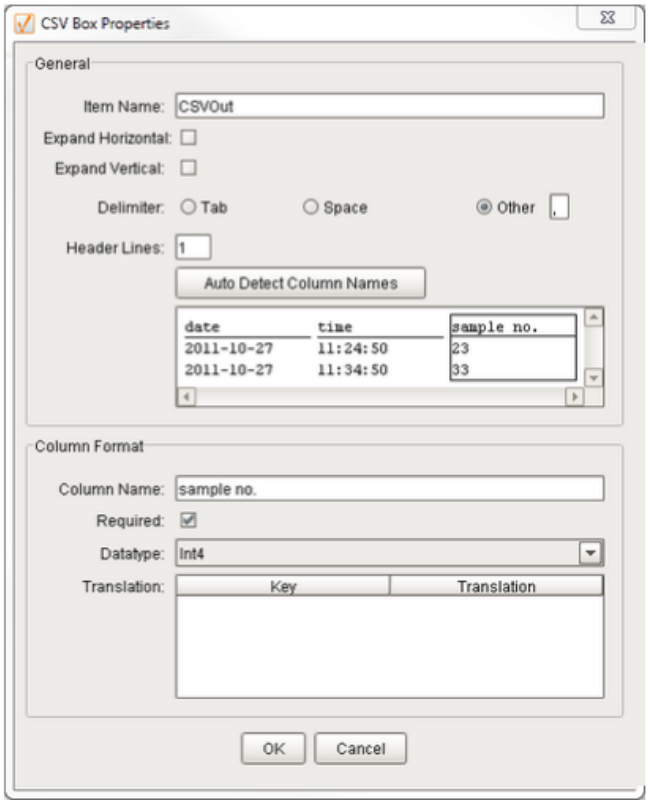
Will parse all data in the columns in a fashion similar to a CSV file. Rows of data contain repeating items.

For example:

date,time,sample no.

2011-10-27,11:24:50,23

2011-10-27,11:34:50,33



CSV Column Parsing Box

CSV Row Parsing Box

Will parse all data in the rows in a fashion similar to a CSV file. A group of rows will be repeated.

For example:

date,2011-10-27

time,11:24:50

sample no.,31

date,2011-10-27

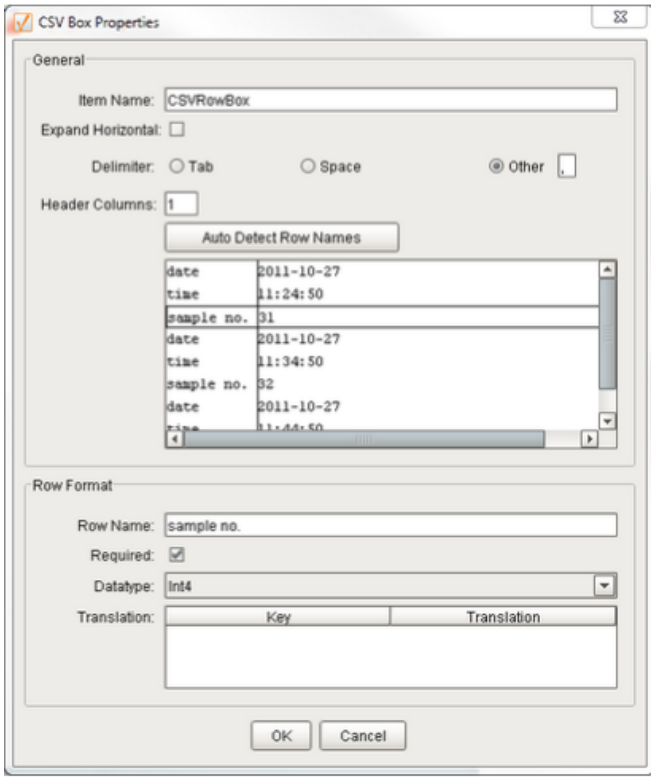
time,11:34:50

sample no.,32

date,2011-10-27

time,11:44:50

sample no.,33



CSV Row Parsing Box

Edit Parsing Box Properties

This will bring up the appropriate editor for the selected parsing box.

Remove Parsing Box

The selected parsing box will be removed.

T oggle Character Grid

A visible grid can be displayed to show the position of all characters.

Parse and Preview Template

This will display a window showing the actual output of the template text after it has been parsed.

Fixed Position Parsing Box

Label Parsing Box

CSV Column Parsing Box

At the core of the Instrument Interface Module is a powerful parsing engine. Beyond processing raw data from serial or text files, any textual data that can be read into either the Ignition client or server can be parsed into meaningful values. This opens up the door to collect data from a variety of sources in the most straightforward manner.

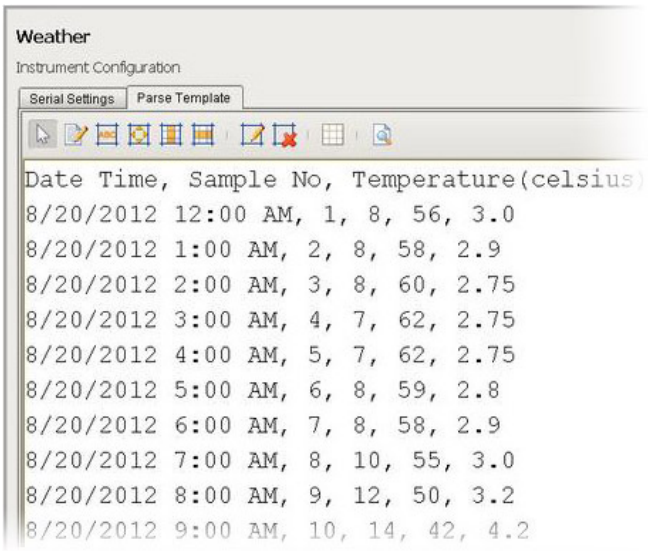
An example is reading temperature and humidity from a device that exposes readings on a simple HTML web page. By using scripts in Ignition, the HTML content can be read and then the temperature and humidity values can be extracted and converted to numeric values using the parsing engine.

The module includes parsing templates that contain textual data with parsing boxes defining the values to be extracted and converted to numeric, date or boolean values.

Other types of parsing boxes allow extracting values at fixed locations, processing CSV columnar data and processing CSV row-based data. A parse template can contain a mixture of any number of the different types of parsing boxes.

CSV Parsing

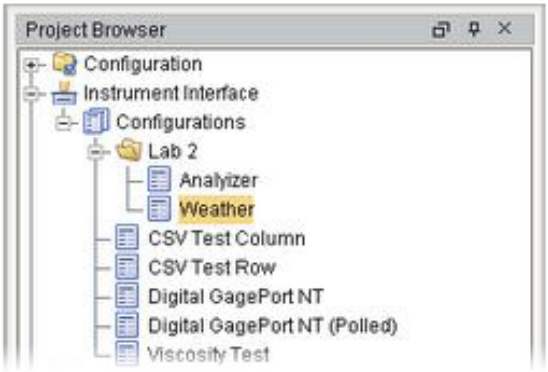
An example of the module's columnar-based CSV parse template that extracts date, time, sample number, temperature and humidity values and makes them available to be accessed in Ignition.



CSV Row Parsing Box

Centralized Instrument Management

Companies have many instruments of the same types, a central location to configure them will reduce the effort to reuse and maintain them. A configuration can be copied or modified to reduce the time required to set up communications with your unique device. The project browser in the Ignition Designer manages all instrument configurations in one central location, facilitating easy set up and maintenance.



8.5.3 Instrument Interface Scripting

This section is a reference for scripting functions provided by the Instrument Interface Module. It also has a reference for any objects that are used by or returned by the scripting functions. The Instrument Interface Module provides scripting functions and has a parsing engine that takes raw data received from an instrument and then extracts the desired values.

Gateway Scripts

Gateway scoped scripts for the instrument interface allows Ignition to parse results without the need for a client to be open.

Client/Designer Scripts

Client and Designer scoped scripts for the instrument interface allows Ignition to parse results coming from the Client computer.

8.5.4 Working with SPC Module

If you want to enter in a sample in the SPC module when you parse a file use the following code for the **onAfterParse** event on the **File Monitor Controller**.

Code Example

```
results = event.getParseResults()
if results.isValid():
locationX = results.getValue("LocationX")
locationY = results.getValue("LocationY")
diameter = results.getValue("Diameter")
```

```
location = "[global]\My Enterprise\Site 1\Packaging\Line 1\Line 1
Quality"
sampleDef = "Measurement"
sample = system.quality.sample.data.getCreateSampleByName('',
sampleDef, location)
sample.setSampleData(1, "LocationX", str(locationX))
sample.setSampleData(1, "LocationY", str(locationY))
sample.setSampleData(1, "Diameter", str(diameter))
sample.setApproved(1)
system.quality.sample.data.updateSample(location, sample, 1)
```

If you want to display the results on a window take a look at the **Sample Entry > Auto Collect** window in the quality demo project. The **onAfterParse** script is slightly different:

Code Example

```
results = event.getParseResults()
if results.isValid():
location = "[global]\My Enterprise\Site 1\Packaging\Line 1\Line 1
Quality"
sampleDef = "Measurement"
sample = system.quality.sample.data.getCreateSampleByName('',
sampleDef, location)
event.source.parent.getComponent("Sample Entry").sample = sample
valueMap = results.createValueMap()
event.source.parent.getComponent("Sample Entry").
populateMeasurements(valueMap)
```

8.6 Web Services

New to Web Services?

Download and test drive this most powerful MES solution available anywhere!

Download and Install Module

A Simple Workflow

Step 1. Database Connection

Step 2. Configuring MES Databases

Step 3. Installing the Production Simulator

Step 4. Web Service Configuration

Web Services Module in a nutshell

Click on the topic you would like to learn more about ...

- Objects
- Scripting

Click here to see Knowledge base articles of **Web Services**.

Info

Sepasoft Web Serices module uses system.ws functions for scripting.

Read this section about licensing...

Licensing and Activation

8.6.1 Web Services Module

The Sepasoft Web Services Module empowers MES software to communicate with other systems using web services over the web or private network. Easily configure web service operations and data types visually, then invoke web service operations from the HMI, SCADA or MES system to read data from or write data to ERP or any other system that supports web services. It shares information that is detailed as work orders, schedules, product definitions, and asset information, or as simple as the weather forecast.

Web service is a form of service oriented architecture (SOA), intended to enable developers to create components that can be assembled and deployed in a distributed and heterogeneous environment. Ignition uses this technology to communicate with other systems. The following sections provide a brief introduction to the technologies used that make up web services.

Web service technologies:

- WSDL
- XML

- SOAP

> **Info**
>
> In the Ignition designer you can see **MES Module Help** while clicking on Help tab. Launching the MES user manual in a web browser is possible through this. This will be disabled if there exist only the Web Services module and no other module is installed.

8.6.2 Intro to Web Services

Web services can be used to retrieve production orders from the ERP systems, in other words ERP systems act as a web service provider. The following sections provide a brief introduction to the technologies used that make up web services. There are plenty of resources provided by the computing industry if you wish to continue to learn more.

What Are Web Services

The W3C (World Wide Web Consortium) defines a Web service as a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

There are two major classes of Web services:

- REST (Representational state transfer)-compliant Web services, in which the primary purpose of the service is to manipulate XML representations of Web resources using a uniform set of stateless operations.
- Arbitrary Web services, in which the service may expose an arbitrary set of operations.

Currently only Arbitrary Web Services implemented via SOAP (Simple Object Access Protocol) are supported in this release.

Network Communications

A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response. Because all communication is in XML, web services are not tied to any one operating system or programming language--Java can talk with Perl; Windows applications can talk with Unix applications.

Web Services are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains. These applications can be local, distributed, or Web-based. Web services are built on top of open standards such as TCP/IP, HTTP, Java, HTML, and XML.

Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction. These systems can include programs, objects, messages, or documents.

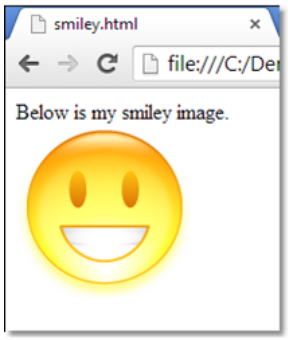
XML

XML is just a standard to textually represent data and is used for all web services. If two systems are sending date values to each other, and one system is using the ISO 8601 standard and the other system only understands RFC 5322, then the date values will be incorrect. This is like two people talking to each other and one is speaking English and the other is speaking French. There will be some misunderstandings.

XML also supports organizing data into records as you can see in the XML sample below. There are multiple material items each having name, category and allergent values.

<table>
<tr><th>HTML</th><th>XML</th></tr>
<tr><td><html>
<div>
Below is my smiley image.
</div>

</html></td><td><xml>
<material>
<name>Almond</name>
<category>Nuts</category>
<allergent>no</allergent>
</material>
<material>
<name>Walnut</name>
<category>Nuts</category>
<allergent>yes</allergent>
</material>
</xml></td></tr>
<tr><td></td><td><table><tr><th>Name</th><th>Category</th><th>Allergent</th></tr><tr><td>Almond</td><td>Nuts</td><td>no</td></tr></table></td></tr>
</table>

HTML	XML
 WS_XML1	<table><tr><th>Name</th><th>Category</th><th>Allergent</th></tr><tr><td>Walnut</td><td>Nuts</td><td>yes</td></tr></table>

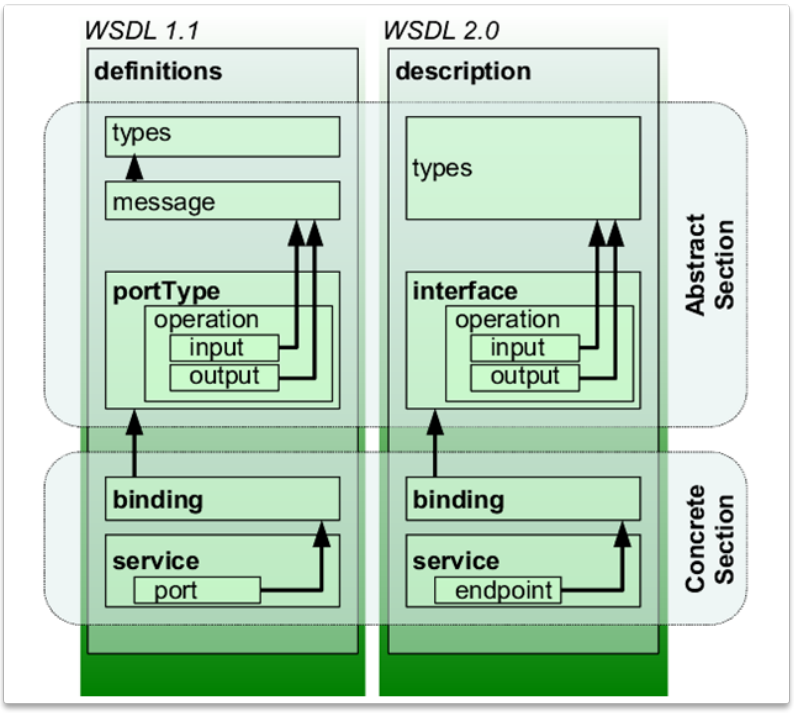
WSDL

WSDL stands for Web Service Description Language and as the name suggests, describes information about the specific web service. This includes, the operations (functions or methods) that are available and data types.

It is used by the Web Services Module to find out information about the web service provided by another system. The only way to know what operations are available and what data types to use is to read the WSDL file from the other system. Once we have the details, then the Web Services Module can show the appropriate setting options.

WSDL Contents

The data types can be simple or complex. Simple data types are the basic types like integer, string, float, etc. Complex data types are just a collection of simple data types or another complex data type.



WS_WSDL1

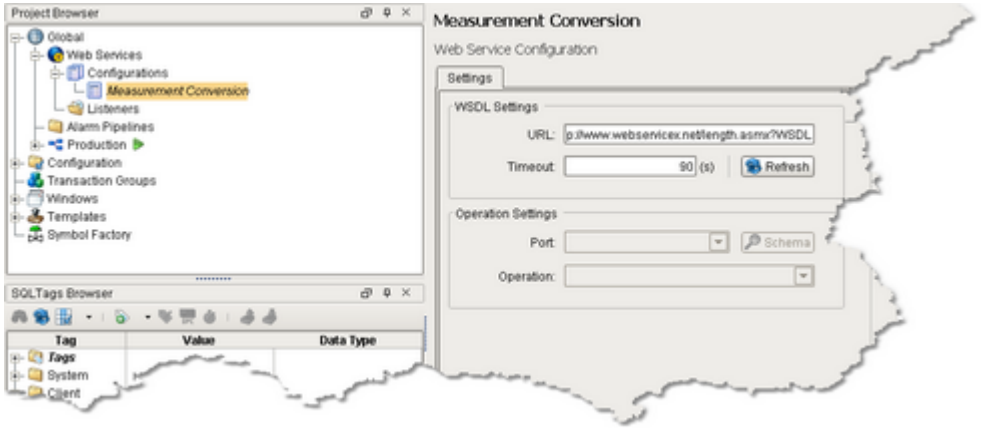
SOAP

SOAP, originally defined as Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of web services in computer networks. It relies on XML information set for its message format, and usually relies on other application layer protocols, most notably Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.

In MES, whenever information is required, the application client invokes remote functions on the server to send and receive data using SOAP.

8.6.3 Web Service Configuration

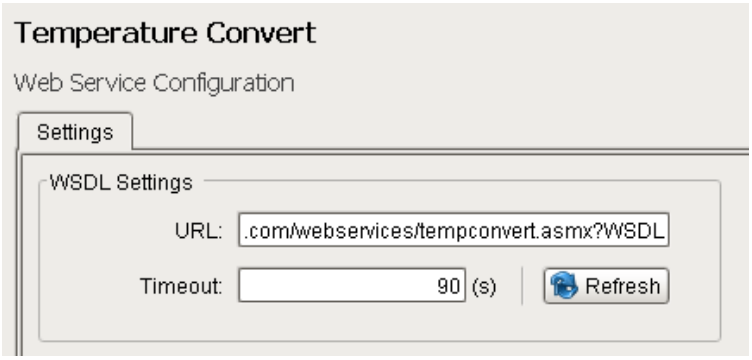
Web Service configuration is where you define such things as the URL for the WSDL and manage the operation provided by the service.



Web Services Configuration

WSDL Settings

This area configures the URL for a web service.



Web Services WSDL Settings

URL

This is the full URL that is required to access the Web Service WSDL.

Examples:

http://www.w3schools.com/webservices/tempconvert.asmx?WSDL

http://www.webservicex.net/length.asmx?WSDL

Timeout

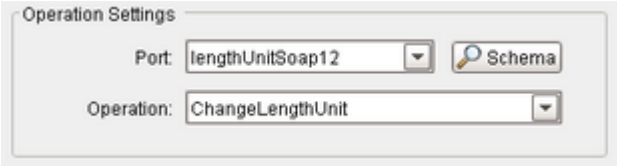
This is the time the module will wait until a response is received from the URL. If no response is received then an error is generated.

Refresh button

This will attempt to connect and process the WSDL file from the entered URL.

Operation Settings

After a valid WSDL Setting has been configured this area allows you to select the available Operations this service supports. The Web Services module internally processes the WSDL and presents the possible selections.



WS_OperationSettings

Web Services Operation Settings

Port

This is the available address or connection point for the web service. It is normally a simple string. Select an available port to allow the available Operations to be populated.

Operation

These are the available actions this web service and port supports. The operation is similar to a method or function call in a traditional programming language.

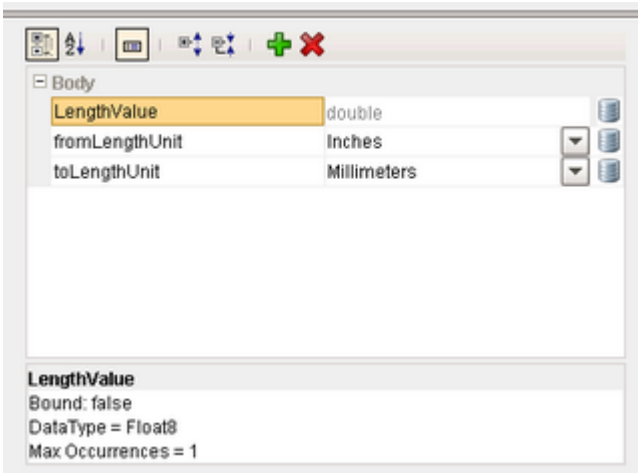
Schema

This will allow you to view the schema used for the selected operation. It is a readable version of the settings contained in the WSDL and allows you to cycle through the ports, operations and body contents of the input and output for the operation.

Parameters

A web service operation will most likely require parameters to be set before the web service is called to run. The parameter section allows you to see those parameters, their data type and optionally set the parameters to a constant value or bind them to a tag.

Some parameters have limited values that can be used and these may be supplied by the operation. These values will appear as a drop down box.



WS_ParameterConfig

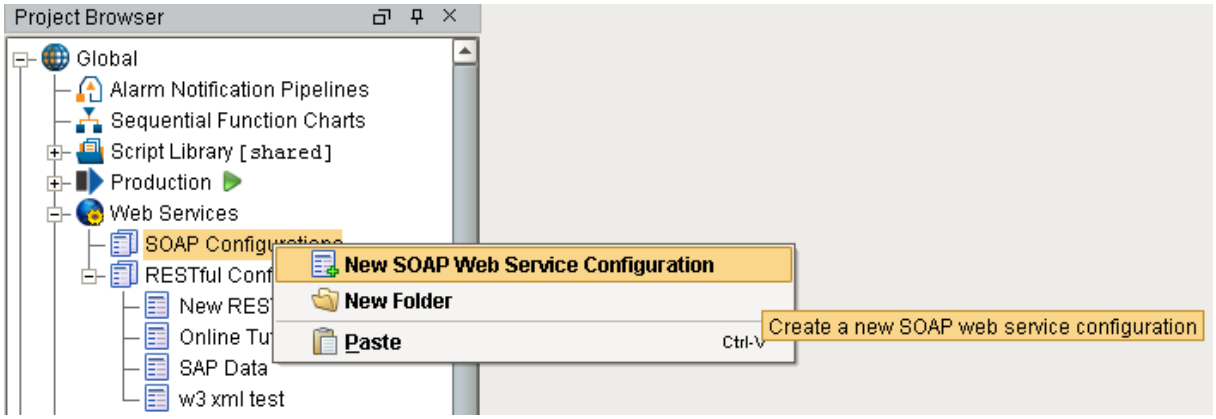
Web Services Parameter Settings

You may need to use these parameter names in scripting calls when running this web service.

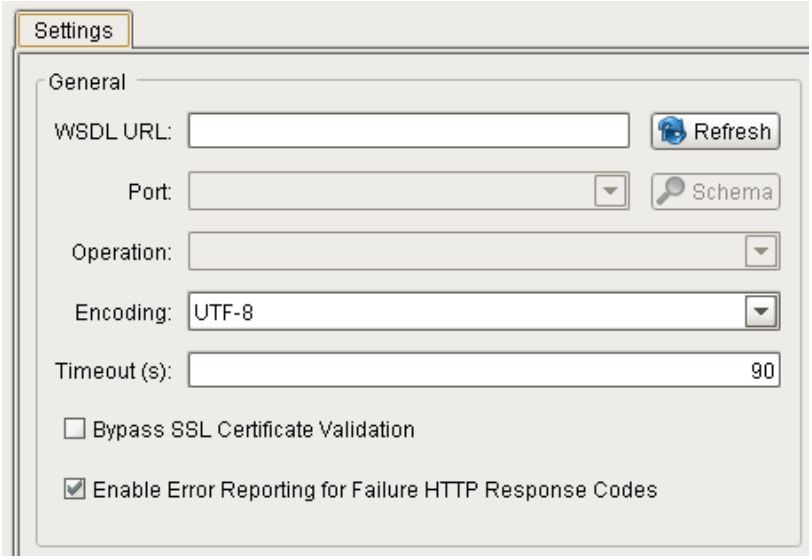
8.6.4 SOAP Configuration

Create new SOAP Configuration

In the Project Browser of the designer, right--click the SOAP Configuration to create a new SOAP configuration.



SOAP Settings



WSDL URL

Port

Operation

Encoding

The available encoding types are UTF_8, UTF_16, ISO_8859_1, Windows_1252, and ASCII.

Timeout

This is the time the module will wait until a response is received from the URL. If no response is received then an error is generated.

Bypass SSL Certificate Validation

The system will bypass SSL certificate validation if the checkbox is in the on state.

Enable Error Reporting for Failure HTTP Response Codes

An exception will be thrown when the web service call results in a code like 404 or other error codes, if this is selected.

HTTP Authentication

Authentication Type

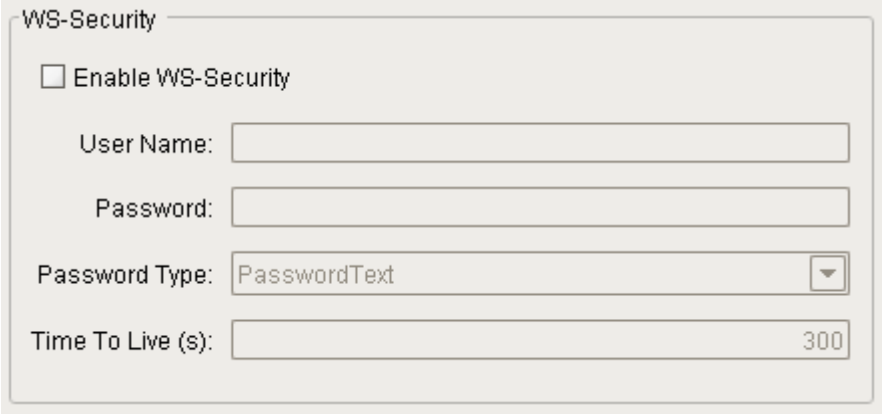
The options are HTTP Basic, HTTP Digest, and HTTP NTLM.

User Name

The user name to set for the new SOAP configuration.

Password

The password to set for the new SOAP configuration.



WS-Security

Enable WS-Security

The system will enable WS-Security if the checkbox is in the on state.

User Name

The user name to set for the WS-Security.

Password

The password to set for the WS-Security.

Password Type

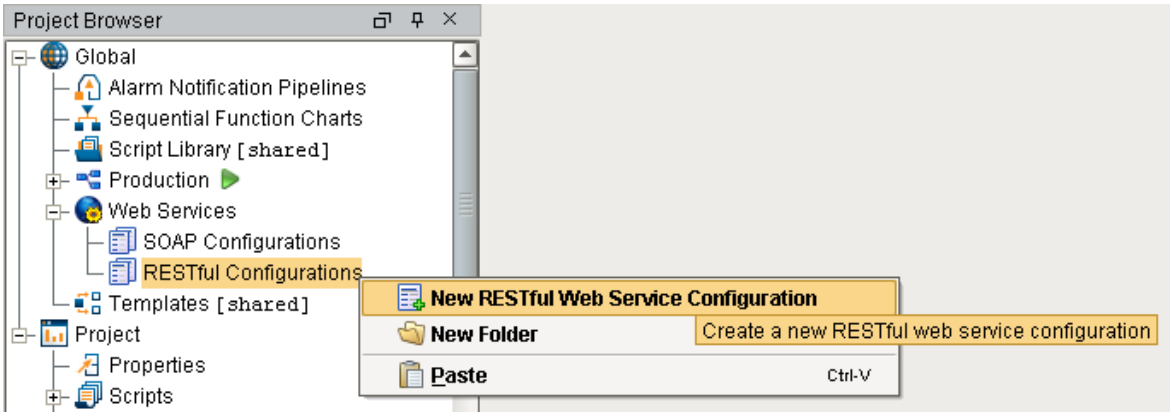
The available password types are PasswordText, PasswordDigest.

Time To Live (s)

8.6.5 RESTful Web Service Configuration

Create new RESTful Configuration

In the Project Browser of the designer, right--click the RESTful Configuration to create a new RESTful configuration.



REST Settings

URL

The target location URL for the RESTful configuration.

Example:

http://demo.sepasoft.com/main/system/webdev/WebServicesAPI/getProductionResults

HTTP Method

HTTP requests include a method, which is a keyword explaining the action that the client wants the server to perform for the material included in the request. The available HTTP methods are GET, POST, PUT, DELETE.

GET

The GET method is used to retrieve information from the given server. Requests using GET should only retrieve data and should have no other effect on the data.

POST

A POST request is used to send data to the server using HTML forms.

PUT

Replaces all current representations of the target resource with the uploaded content.

DELETE

Removes all current representations of the target resource.

Data Format

The available options are JSON and XML.

Encoding

The available encoding types are UTF_8, UTF_16, ISO_8859_1, Windows_1252, and ASCII.

Timeout

This is the time the module will wait until a response is received from the URL. If no response is received then an error is generated.

Max Retries

The maximum number of retries, in case of connection failure. Default value is 3.

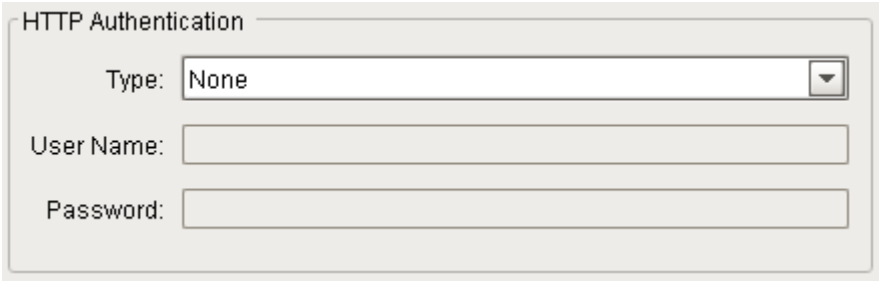
Bypass SSL Certificate Validation

The system will bypass SSL certificate validation if the checkbox is in the on state.

Enable Error Reporting for Failure HTTP Response Codes

An exception will be thrown when the web service call results in a code like 404 or other error codes, if this is selected.

HTTP Authentication



Authentication

The options are HTTP Basic, HTTP Digest, and HTTP NTLM.

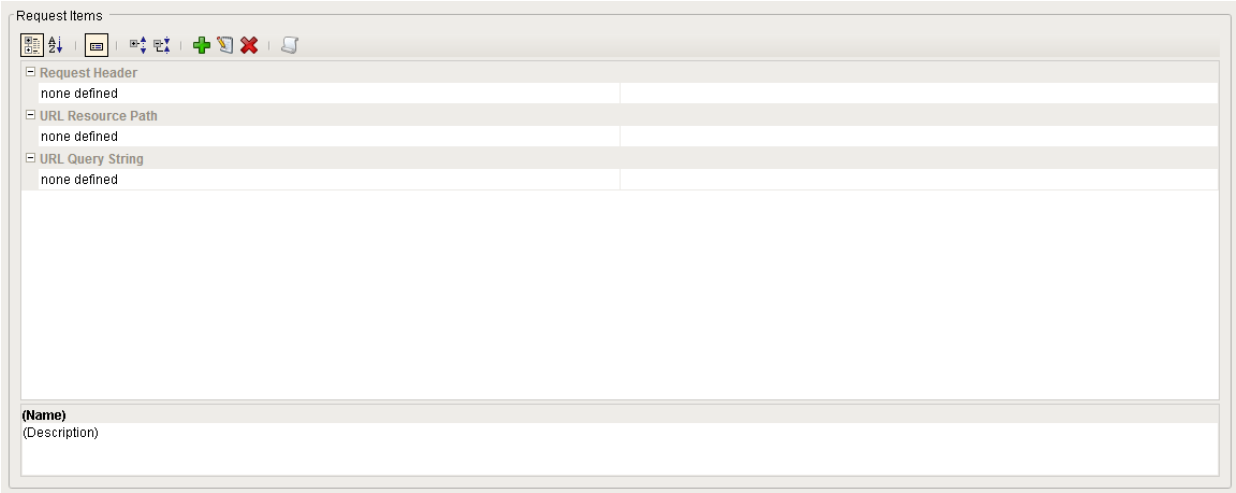
User Name

The user name to set for the new RESTful configuration.

Password

The password to set for the new RESTful configuration.

Request Items



Request Headers

Depending on the Web Service, these can be used to identify the format of the body object you are POSTing to the web service, or perhaps an API key or other identifying tool.

URL Resource Path

These are additional parts of the URL, divided by slashes. For example:

> http://demo.sepasoft.com/main/system/webdev/WebServicesAPI/getProductCodeList?filter=SA

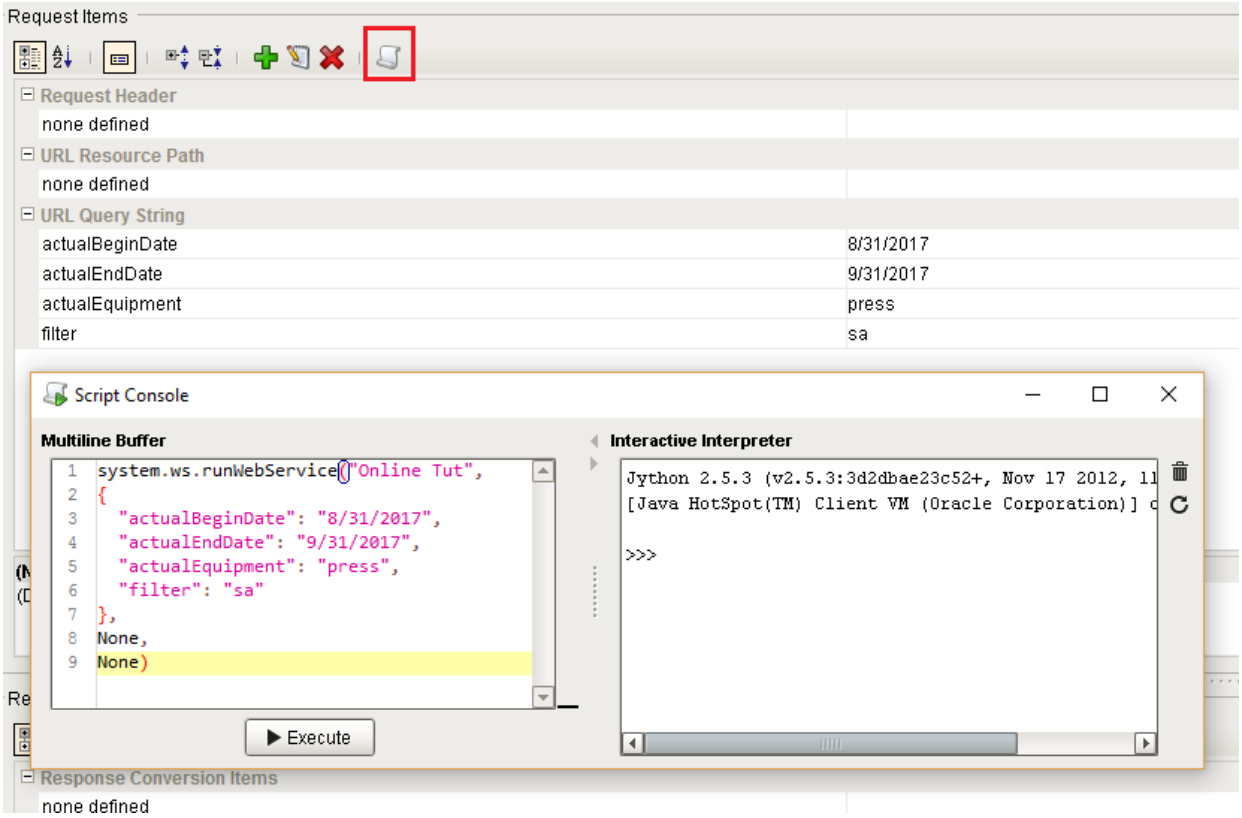
In this example, "/main", "/system", "/webdev", "/WebServicesAPI", and "/getProductCodeList" are all resource paths.

URL Query String

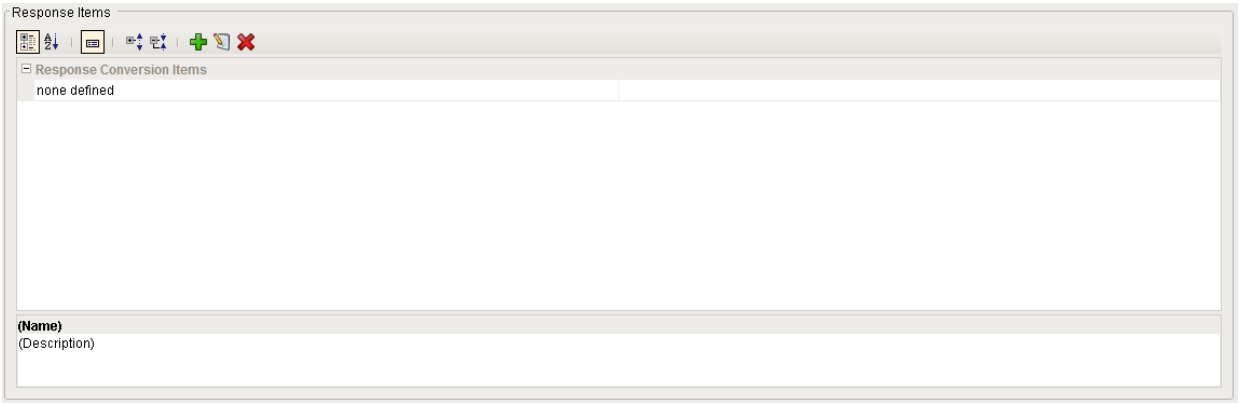
These follow any resource paths, and follow a question mark. They have a key and a value. Using the above example, "filter=SA" is a query string.

New Feature

Click on  for copying a completed 'runWebService' scripting function to the clipboard.



Response Items



8.7 Barcode Scanner

The Sepasoft Barcode scanner module is a utility module. With Sepasoft Barcode scanner, customers can use barcode scanning capabilities to facilitate material handling. User can scan badge to sign off on materials, Master Production Records, Batch Production Records, and other production tasks. Software also includes ability to hide manufacturing records not in use and added option for customers to include their own logos.

> **Info**
>
> Sepasoft Barcode Scanner module uses system.barcode.scanner functions for scripting.

8.7.1 Barcode Scanner Module Overview

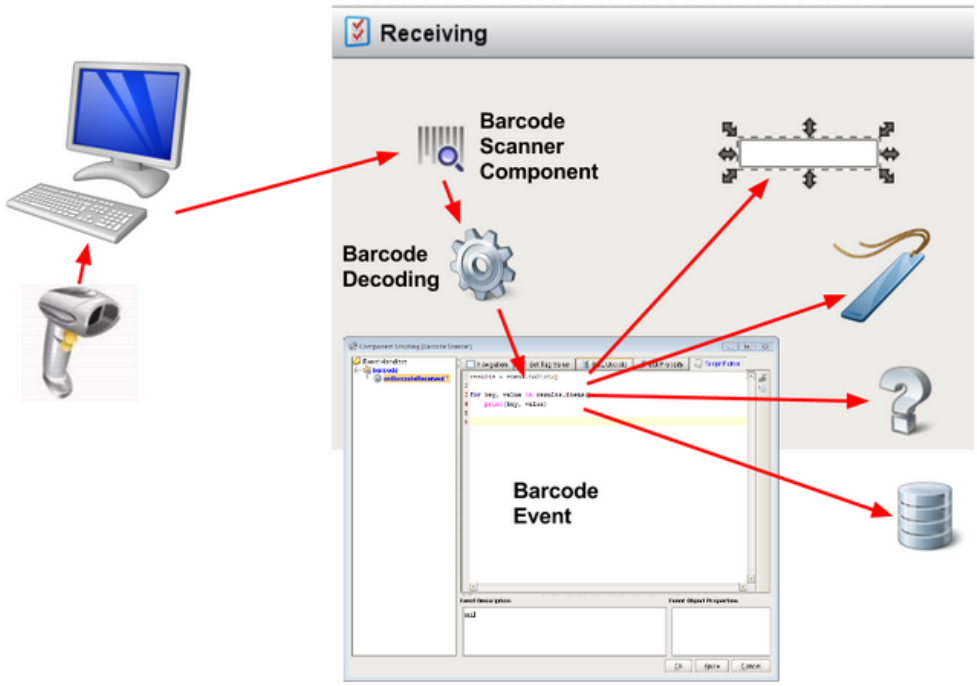
Barcode Scanner module extends Ignition to provide the ability to detect and decode barcode input. The module provides both a controller to be used on the client and gateway scripting methods for server barcode decoding. The module works with traditional one dimensional (1D) barcode formats, plus the newer two dimensional (2D) and GS1 international standard of formats that can contain multiple pieces of data.

 SCRN_Sample_UPC UPC Barcode	 SCRN_Sample_Code128 Code 128	 SCRN_Sample_DataMatrix Data Matrix

The module comes pre-configured with 100+ barcode patterns to decode standard barcodes like UPC, EAN, and GTIN, plus the full range of the GS1 application identification (AI) standard formats for trade items, logistic units, assets, locations, service relationships, and special applications. The pre-configured patterns will meet the needs of most operations using standard barcode formats and content. In addition, the module can be fully customized to read proprietary or industry specific barcode content.

Commonly reading barcode input would follow the scenario of having an operator place the cursor on a select input field, then scan the barcode to input the barcode's content into the selected field. If multiple barcodes needed to be scanned or the content of the barcode contained multiple pieces of data, then it often requires additional intervention by the operator to get the input recorded correctly.

With the BarcodeScanner component, the scenario above is greatly improved while reducing the risk of input mistakes. Once the Barcode Scanner component is placed on an existing screen, it will listen for barcode input via a keyboard wedge. This is done in the background and is independent of the focused input component. Once it detects barcode input, it then decodes the barcode based on one of more barcode patterns and raises a script event with the results. The script can then simply put the results into the correct input field(s), update tag(s), write to database table(s), etc.



SCRN_Component2

Barcode Detection and Decoding

Ignition supports Barcode entries and is using barcodes on most reports to identify the connected MES objects. As most of our other features, the barcode capability is already integrated in the Sepasoft MES software. Barcodes are displayed on reports, and you can define your own custom barcode labels that will show product specific information and custom images. In short, it provides intuitive user entry and reporting. Customers can use barcode scanning capabilities to facilitate material handling. User can scan badge to sign off on materials, Master Production Records, Batch Production Records, and other production tasks. As soon as we scan an item, the corresponding MES object opens.

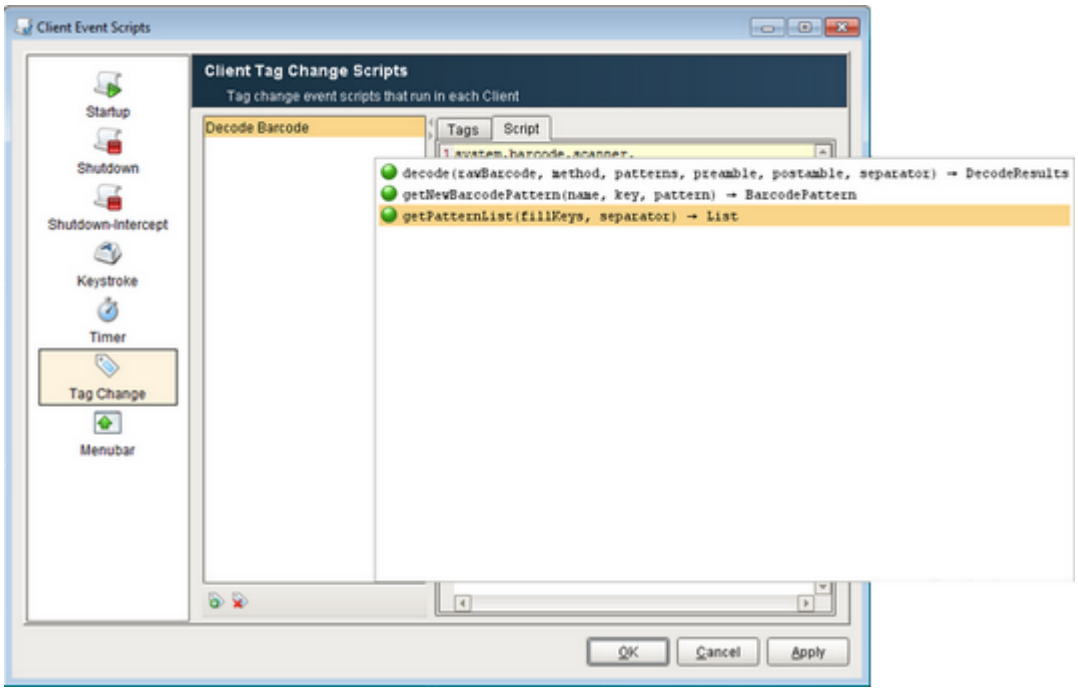
The probability of occurrence of human error is reduced drastically through a barcode scanner. Barcode scan also reduce the time consumption for the manual data entry. Inventory control is enhanced since it is fast and reliable. Barcodes can be attached to products and equipments. Data obtained from barcode scanners may be used for tracking the resources.

Also you can create multiple designs and manage the label designs using revision management. This is an additional benefit of Sepasoft MES Barcode scanner component.

8.7.2 Client / Gateway Scripts-BarcodeScanner

The functionality of the BarcodeScanner component is also available for gateway and client scripting. The following methods can be utilized in a gateway script to create regular expressions (regex) patterns and decode a barcode by searching the raw barcode for the list of patterns. The results of the barcode decoding can then be used by in the gateway script.

In the Ignition script editor, documentation for the script functions can be accessed by pressing control-space after typing in '**system.**'. For all the track and trace script functions, type in **system.barcode.** and press control-space to see the associated function and documentation.



Scanner_AutoComplete1

Ignition Script Auto Document Feature

Barcode decoding uses Regex patterns to decode raw barcode data. The Barcode Scanner Module is pre-configured with a set of 100+ patterns to include common and GS1 Application Identifier standards. If the default patterns do not fit the required needs, then custom patterns can be defined. Below is the list of predefined patterns:

NAME	KEY	REGEX PATTERN
Default	Default	"(.*)"
GTIN-14	GTIN-14	"^(\\d{14})$

NAME	KEY	REGEX PATTERN
GTIN-12 UPC	GTIN-12	"^(\\d{12})$
GTIN-13 EAN	GTIN-13	"^(\\d{13})$
GTIN-8 EAN	GTIN-8	"^(\\d{8})$
SSCC	GS1-00	"(00)(\\d{18})"
GTIN	GS1-01	"(01)(\\d{14})"
CONTENT	GS1-02	"(02)(\\d{14})"
BATCH/LOT	GS1-10	"(10)(\\d{1,20}" + separator + "|\\d{1,20}$)"
PROD DATE	GS1-11	"(11)(\\d{6})"
DUE DATE	GS1-12	"(12)(\\d{6})"
PACK DATE	GS1-13	"(13)(\\d{6})"
BEST BY	GS1-15	"(15)(\\d{6})"
SELL BY	GS1-16	"(16)(\\d{6})"
USE BY	GS1-17	"(17)(\\d{6})"

NAME	KEY	REGEX PATTERN
VARIANT	GS1-20	"(20)(\\d{2})"
SERIAL	GS1-21	"(21)(.{1,20}" + separator + "|.{1,20}$)"
ADDITIONAL ID	GS1-240	"(240)(.{1,30}" + separator + "|.{1,30}$)"
CUST.PART NO.	GS1-241	"(241)(.{1,30}" + separator + "|.{1,30}$)"
MTO VARIANT	GS1-242	"(242)(\\d{1,6}" + separator + "|\\d{1,6}$)"
PCN	GS1-243	"(243)(.{1,20}" + separator + "|.{1,20}$)"
SECONDARY SERIAL	GS1-250	"(250)(.{1,30}" + separator + "|.{1,30}$)"
REF. TO SOURCE	GS1-251	"(251)(.{1,30}" + separator + "|.{1,30}$)"
GDTI	GS1-253	"(253)(\\d{13}.{1,17}" + separator + "|\\d{13}.{1,17}$)"
GLN EXTENSIION COMP	GS1-254	"(254)(.{1,20}" + separator + "|.{1,20}$)"
GCN	GS1-255	"(255)(\\d{13}\\d{1,12}" + separator + "|\\d{13}\\d{1,12}$)"
VAR. COUNT	GS1-30	"(30)(\\d{1,8}" + separator + "|\\d{1,8}$)"
NET WEIGHT (kg)		"(310)([0-6]{1})(\\d{6})"

NAME	KEY	REGEX PATTERN
	GS1-310	
LENGTH (m)	GS1-311	"(311)([0-6]{1})(\\d{6})"
WIDTH (m)	GS1-312	"(312)([0-6]{1})(\\d{6})"
HEIGHT (m)	GS1-313	"(313)([0-6]{1})(\\d{6})"
AREA (m^2)	GS1-314	"(314)([0-6]{1})(\\d{6})"
NET VOLUME (l)	GS1-315	"(315)([0-6]{1})(\\d{6})"
NET VOLUME (m^3)	GS1-316	"(316)([0-6]{1})(\\d{6})"
NET WEIGHT (lb)	GS1-320	"(320)([0-6]{1})(\\d{6})"
LENGTH (i)	GS1-321	"(321)([0-6]{1})(\\d{6})"
LENGTH (f)	GS1-322	"(322)([0-6]{1})(\\d{6})"
LENGTH (y)	GS1-323	"(323)([0-6]{1})(\\d{6})"
WIDTH (i)	GS1-324	"(324)([0-6]{1})(\\d{6})"
WIDTH (f)		"(325)([0-6]{1})(\\d{6})"

NAME	KEY	REGEX PATTERN
	GS1-325	
WIDTH (y)	GS1-326	"(326)([0-6]{1})(\\d{6})"
HEIGHT (i)	GS1-327	"(327)([0-6]{1})(\\d{6})"
HEIGHT (f)	GS1-328	"(328)([0-6]{1})(\\d{6})"
HEIGHT (y)	GS1-329	"(329)([0-6]{1})(\\d{6})"
GROSS WEIGHT (kg)	GS1-330	"(330)([0-6]{1})(\\d{6})"
LENGTH (m),log	GS1-331	"(331)([0-6]{1})(\\d{6})"
WIDTH (m),log	GS1-332	"(332)([0-6]{1})(\\d{6})"
HEIGHT (m),log	GS1-333	"(333)([0-6]{1})(\\d{6})"
AREA (m^2),log	GS1-334	"(334)([0-6]{1})(\\d{6})"
VOLUME (l),log	GS1-335	"(335)([0-6]{1})(\\d{6})"
VOLUME (m^3),log	GS1-336	"(336)([0-6]{1})(\\d{6})"
KG PER m^2		"(337)([0-6]{1})(\\d{6})"

NAME	KEY	REGEX PATTERN
	GS1-337	
GROSS WEIGHT (lb)	GS1-340	"(340)([0-6]{1})(\\d{6})"
LENGTH (i),log	GS1-341	"(341)([0-6]{1})(\\d{6})"
LENGTH (f),log	GS1-342	"(342)([0-6]{1})(\\d{6})"
LENGTH (y),log	GS1-343	"(343)([0-6]{1})(\\d{6})"
WIDTH (i),log	GS1-344	"(344)([0-6]{1})(\\d{6})"
WIDTH (f),log	GS1-345	"(345)([0-6]{1})(\\d{6})"
WIDTH (y),log	GS1-346	"(346)([0-6]{1})(\\d{6})"
HEIGHT (i),log	GS1-347	"(347)([0-6]{1})(\\d{6})"
HEIGHT (f),log	GS1-348	"(348)([0-6]{1})(\\d{6})"
HEIGHT (y),log	GS1-349	"(349)([0-6]{1})(\\d{6})"
AREA (i^2)	GS1-350	"(350)([0-6]{1})(\\d{6})"
AREA (f^2)		"(351)([0-6]{1})(\\d{6})"

NAME	KEY	REGEX PATTERN
	GS1-351	
AREA (y²)	GS1-352	"(352)([0-6]{1})(\\d{6})"
AREA (i²),log	GS1-353	"(353)([0-6]{1})(\\d{6})"
AREA (f²),log	GS1-354	"(354)([0-6]{1})(\\d{6})"
AREA (y²),log	GS1-355	"(355)([0-6]{1})(\\d{6})"
NET WEIGHT (t)	GS1-356	"(356)([0-6]{1})(\\d{6})"
NET VOLUME (oz)	GS1-357	"(357)([0-6]{1})(\\d{6})"
NET VOLUME (q)	GS1-360	"(360)([0-6]{1})(\\d{6})"
NET VOLUME (g)	GS1-361	"(361)([0-6]{1})(\\d{6})"
VOLUME (q),log	GS1-362	"(362)([0-6]{1})(\\d{6})"
VOLUME (g),log	GS1-363	"(363)([0-6]{1})(\\d{6})"
VOLUME (i³)	GS1-364	"(364)([0-6]{1})(\\d{6})"
VOLUME (f³)		"(365)([0-6]{1})(\\d{6})"

NAME	KEY	REGEX PATTERN
	GS1-365	
VOLUME (y³)	GS1-366	"(366)([0-6]{1})(\\d{6})"
VOLUME (i³),log	GS1-367	"(367)([0-6]{1})(\\d{6})"
VOLUME (f³),log	GS1-368	"(368)([0-6]{1})(\\d{6})"
VOLUME (y³),log	GS1-369	"(369)([0-6]{1})(\\d{6})
COUNT	GS1-37	"(37)(\\d{1,8}" + separator + "|\\d{1,8}$)"
AMOUNT	GS1-390	"(390)([0-9]{1})(\\d{1,15}" + separator + "|\\d{1,15}$)"
AMOUNT	GS1-391	"(391)([0-9]{1})(\\d{3})(\\d{1,18}" + separator + "|\\d {1,18}$)"
PRICE	GS1-392	"(392)([0-9]{1})(\\d{1,15}" + separator + "|\\d{1,15}$)"
PRICE	GS1-393	"(393)([0-9]{1})(\\d{3})(\\d{1,18}" + separator + "|\\d {1,18}$)"
ORDER NUMBER	GS1-400	"(400)(.{1,30}" + separator + "|.{1,30}$)"
GINC	GS1-401	"(401)(.{1,30}" + separator + "|.{1,30}$)"
GSIN		"(402)(\\d{17}" + separator + "|\\d{17}$)"

NAME	KEY	REGEX PATTERN
	GS1-402	
ROUTE	GS1-403	"(403)(.{1,30}" + separator + "|.{1,30}$)"
SHIP TO LOC	GS1-410	"(410)(\\d{13})"
BILL TO	GS1-411	"(411)(\\d{13})"
PURCHASE FROM	GS1-412	"(412)(\\d{13})"
SHIP FOR LOC	GS1-413	"(413)(\\d{13})"
LOC No	GS1-414	"(414)(\\d{13})"
PAY TO	GS1-415	"(415)(\\d{13})"
SHIP TO POST	GS1-420	"(420)(.{1,20}" + separator + "|.{1,20}$)"
SHIP TO POST	GS1-421	"(421)(\\d{3})(.{1,20}" + separator + "|.{1,20}$)"
ORIGIN	GS1-422	"(422)(\\d{3})"
COUNTRY - INIT PROCESS	GS1-423	"(423)(\\d{3})(\\d{3,12}" + separator + "|\\d{3,12}$)"
COUNTRY - PROCESS		"(424)(\\d{3})"

NAME	KEY	REGEX PATTERN
	GS1-424	
COUNTRY - DISASSEMBLY	GS1-425	"(425)(\\d{3})"
COUNTRY - FULL PROCESS	GS1-426	"(426)(\\d{3})"
ORIGIN SUBDIVISION	GS1-427	"(427)(.{1,3}" + separator + "|.{1,3}$)"

Sample Gateway Tag Change Scripts

Example 1

```
# Gateway Tag Change Script to decode a UPC, EAN, GTIN barcode

# Setup the logger to see activity on the Ignition Gateway console
# or just use print statements and look at wrapper.log file.
from org.apache.log4j import Logger
log = Logger.getLogger('GTINDecoder')

# Import java needed classes
from java.util import List

# Only execute if not initializing and tag value is not null
if (initialChange == 0 and newValue.value != None):
 # Get rawBarcode from tag value
 rawBarcode = newValue.value

 # Get only the predefined regex patterns for GTIN-12 UPC, GTIN-
13 EAN, & GTIN-14
 patterns = system.barcode.scanner.getPatternList("GTIN-12,GTIN-
13,GTIN-14", "")

 # Call the decode method for a single pass search with a
preamble of \u0002 ascii STX (start of text),
 # and no postamble, or separator in barcode
 results = system.barcode.scanner.decode(rawBarcode, "SinglePass",
patterns, u"\u0002", "", "")
```

```
 if (results.hasErrorMessage()):
        # Log error message
        log.info("Error from barcode scan: " + results.
getErrorMessage())
 else:
        # Get a Python dictionary of the decoding results
        resultsDict = results.toDict()

        # Loop through results
        for key,value in resultsDict.items():
                log.info("Key: " + key + ", Value: " + value.
toString())
```

Example 2

```
# Gateway Tag Change Script to decode a GS1 Active Matrix barcode

# Setup the logger to see activity on the Ignition Gateway console
# or just use print statements and look at wrapper.log file.
from org.apache.log4j import Logger
log = Logger.getLogger('GS1Decoder')

# Import java needed classes
from java.util import List

# Only execute if not initializing and tag value is not null
if (initialChange == 0 and newValue.value != None):
 # Get rawBarcode from tag value
 rawBarcode = newValue.value

 # Get the predefined regex patterns for GS1 Application
Identifer (AI) we are interested in, The FNC1
 # separator for the variable length AIs is Unicode \u001d and
will be inserted into the patterns.
 patterns = system.barcode.scanner.getPatternList("GS1-10,GS1-17,
GS1-01,GS1-390,GS1-310", u"\u001d")

 #Uncomment to see patterns in list
 #for p in patterns:
        #log.info("key=%s regex=%s" %(p.getKey(), p.
getRegexPattern()))

 # Call the decode method for a GS1 consume search with a
preamble of ]d1,
 # and postamble of \u001a (LF line feed), and separator of
\u001d (GS Group Separator for FNC1) in barcode
 results = system.barcode.scanner.decode(rawBarcode, "Consume",
patterns, u"]d1", u"\u001a", u"\u000d")
```

```
 if (results.hasErrorMessage()):
         # Log error message
         log.info("Error from barcode scan: " + results.
getErrorMessage())
 if (results.hasUnmatched()):
         # Log unmatched
         log.info("Unmatched: " + results.getUnmatched())

 # Get a Python dictionary of the decoding results
 resultsDict = results.toDict()

 # Loop through results
 for key,value in resultsDict.items():
         log.info("Key: " + key + ", Value: " + value.toString())
```

Example 3

```
# Gateway Tag Change Script to decode a custom regex pattern

# Setup the logger to see activity on the Ignition Gateway console
# or just use print statements and look at wrapper.log file.
from org.apache.log4j import Logger
log = Logger.getLogger('CustomDecoder')

# Import java needed classes
from java.util import List

# Only execute if not initializing and tag value is not null
if (initialChange == 0 and newValue.value != None):
 # Get rawBarcode from tag value
 rawBarcode = newValue.value

 # Create a new empty patterns list
 patterns = []

 # Create customer barcode pattern and add to list
 pattern = system.barcode.scanner.getNewBarcodePattern("My
Pattern Name", "MyPatternKey", u"^(\\d{8})$")
 patterns.append(pattern)

 # Call the decode method for a single pass search with no
preamble, postamble, or separator in barcode
 results = system.barcode.scanner.decode(rawBarcode, "SinglePass",
patterns, "", "", "")

 if (results.hasErrorMessage()):
         # Log error message
         log.info("Error from barcode scan: " + results.
getErrorMessage())
```

```
# Get a Python dictionary of the decoding results
resultDict = results.toDict()

# Loop through results
for key,value in resultDict.items():
        log.info("Key: " + key + ", Value: " + value.toString())
```

8.8 MES Enterprise

MES Enterprise module is a remote Gateway administration system, allowing you to manage Gateways and automate tasks from a single controller. It connects multiple MES Ignition Gateways across your entire enterprise to form a large, centrally managed MES solution. This module analyzes MES data from multiple production facilities at the enterprise server.

8.9 Production Simulator

The Production Simulator Module is a free module that provide a simulator device that you can use with the MES modules. The simulator tags generate values you can use to simulate PLC tags. The tags are values generated can be controlled through CSV files. These files should be copied into **C:\Program Files\Inductive Automation\Ignition\data\drivers** directory so that the device can read them in as tags.

For Linux systems, the file location: " **/var/lib/ignition/data/drivers** "

Every CSV file is for a separate simulator you can work with. The more files you add, the more folders you get in the Simulator folder of the OPC Browser.

8.9.1 How to create CSV files for Production Simulator

You can simulate any real life scenario by creating the CSV files. The example file on the right has two sections: **Cells** and **Events**

Cells Section

This section is optional, where you specify the information about the cell.

Events Section

Events area is where you specify values for all the tags. You can add as many columns of tags you want to bring in. Make sure to put back slashes to organize in folders and specify the name of the tag you want as well as their datatypes.

Table view of an example CSV file

<Cells>						
Name	Upstream Cell	Downstream Cell	Startup Count	Backup State	Infeed Rate /Hr	R V
Filler		Sealer	10	4	3600	1(
Sealer	Filler	Capper	10	4	3600	1(
Capper	Sealer	Labeler	10	4	3600	1(
Labeler	Capper	Inspection	10	4	3600	1(
Inspection	Labeler	Casepacker	10	4	3600	1(
CasePacker	Inspection	Palletizer	10	4	3600	1(
Palletizer	Casepacker		2	4	360	1(
<Events>						
DOW	Time	Filler\State	Sealer\State	Capper\State	Labeler\State	In
Int32	String	Int32	Int32	Int32	Int32	In
1	12:00:00 AM					6

1	12:00:15 AM					1
1	12:00:30 AM	22				
1	12:00:45 AM	1				
1	12:01:00 AM					
1	12:01:15 AM				8	
1	12:01:30 AM				8	
1	12:01:45 AM				1	
1	12:02:00 AM			6		
1	12:02:15 AM			1		
1	12:02:30 AM					
1	12:02:45 AM					
1	12:03:00 AM					
1						

<Cells>						
	12:03:15 AM					
1	12:03:30 AM					
1	12:03:45 AM	8				
1	12:04:00 AM	8				
1	12:04:15 AM	8				
1	12:04:30 AM	1				

9 Appendix A: Reference Guide

The appendix is your reference. Once you become experienced in developing with Sepasoft MES you will most likely keep this page open on your desktop or on your second screen. This page provides you the fastest route to the information you are looking for.

9.1 Components

9.2 MES Objects

9.3 Scripting Functions

Collapse allExpand all Collapse all

9.4 Binding Functions

Collapse allExpand all Collapse all

Collapse allExpand all Collapse all

Collapse allExpand all Collapse all

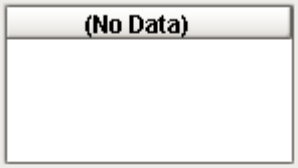
9.5 Components

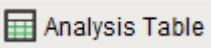
MES has a lot of components built-in to provide user interaction with the MES data. The components vary from one module to other. This is because we introduce them based on their requirement. All system processes are placed into separate components so that all of the data and functions inside each component are semantically related. Components are placed inside the palette, you can easily drop it into the root container of the designer window. There is a Property Editor panel to alter the component's properties, which changes the component's appearance and behavior. To make the component do something useful, like display dynamic information or control a device register, you configure property bindings for the component. In order to make the component react to user interaction, you can configure the event handlers.

9.5.1 Common Components

Analysis Table

General



Component Palette Icon: 

Description

A component that displays tabular data with drill down capabilities. This extends from the Table Component  that comes with Ignition.

Properties

Name	Scripting	Category	Property Type	Description
Row Selection Allowed	rowSelectionAllowed	Behavior	boolean	This flag is used in conjunction with the Column Selection Allowed flag to determine whether not whole-rows, whole-columns, or both.
Column Selection Allowed	columnSelectionAllowed	Behavior	boolean	This flag is used in conjunction with the Row Selection

Name	Scripting	Category	Property Type	Description
				Allowed flag to determine whether not whole-rows, whole-columns, or both.
Allow Export	allowExport	Appearance	boolean	If true, allow user to export data in table.
Header Visible	headerVisible	Appearance	boolean	Controls the visibility of the table's header.
Resizing Allowed	resizingAllowed	Behavior	boolean	Whether or not the user is allowed to resize table headers or not.
Row Height	rowHeight	Appearance	int	The height of each row, in pixels.
Odd Row Background	oddBackground	Appearance	Color	The color which odd rows will be colored if background mode is 'Alternating'.
Selection Background	selectionBackground	Appearance	Color	The background color of a selected cell.
Selection Foreground	selectionForeground	Appearance	Color	The foreground color of a selected cell.
	showHorizontalLines	Appearance	boolean	

Name	Scripting	Category	Property Type	Description
Show Horizontal Grid Lines?				Displays horizontal gridlines making it easier to read.
Show Vertical Grid Lines?	showVerticalLines	Appearance	boolean	Displays vertical gridlines making it easier to read.
Grid Line Color	gridColor	Appearance	Color	The color used to draw grid lines.
Initially Selected Row	initialRowSelection	Behavior	int	The index of the row that should be selected by default when this table's data is filled in. Note that you must save the table with no selection in order for this to work.
Data	data	Data	Dataset	The data for this table.
Drill Down Options	drillDownOptions	Data	DataSet	Dataset with drill down options.
Previous Drill Down Enabled	previousDrillDownEnabled	Data	Boolean	If true, show previous in drill down menu.
Column Attributes Data	columnAttributesData	Data	Dataset	The dataset describing the column attributes.
TestData	test	Misc	boolean	

Name	Scripting	Category	Property Type	Description
				Toggle this property to fill in the table's data with random data.
Selected Column	selectedColumn	Data	int	The index of the first selected column, or -1 if none.
Selected Row	selectedRow	Data	int	The index of the first selected row, or -1 if none.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

cell

cellEdited

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it.

Property	Description
source	The component that fired this event.
oldValue	The old value in the cell that changed.
newValue	The new value in the cell that changed.
row	The row of the dataset this cell represents.
column	The column of the dataset this cell represents.

drillDown

drillDown

Is fired when drill down menu item is selected. Excludes the "Back" menu item.

Property	Description
source	The component that fired this event.
drillDownName	Text of selected drill down option menu item.
category	Value of first column for the selected row.

back

Property	Description
source	The component that fired this event.
drillDownName	Text of selected drill down option menu item.
category	Value of first column for the selected row.

focus

focusGained

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it.

Property	Description
source	The component that fired this event.
oppositeComponent	The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vise versa.

focusLost

This event occurs when a component that had the input focus lost it to another component.

Property	Description
source	The component that fired this event.
oppositeComponent	The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vise versa.

key

keyPressed

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.

Property	Description
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

keyReleased

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
altDown	

Property	Description
	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

keyTyped

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	

Property	Description
	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

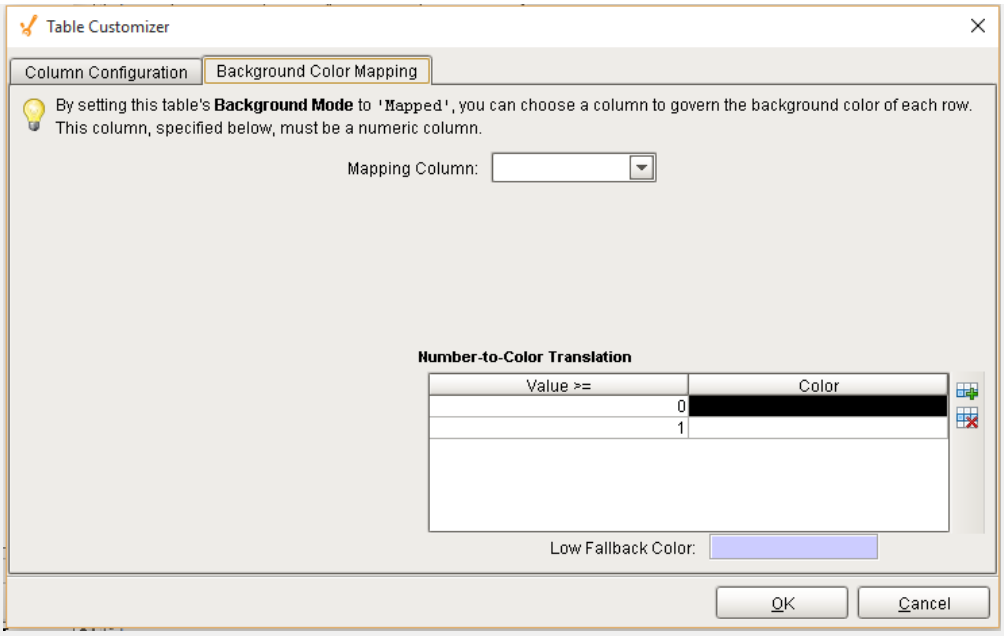
propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

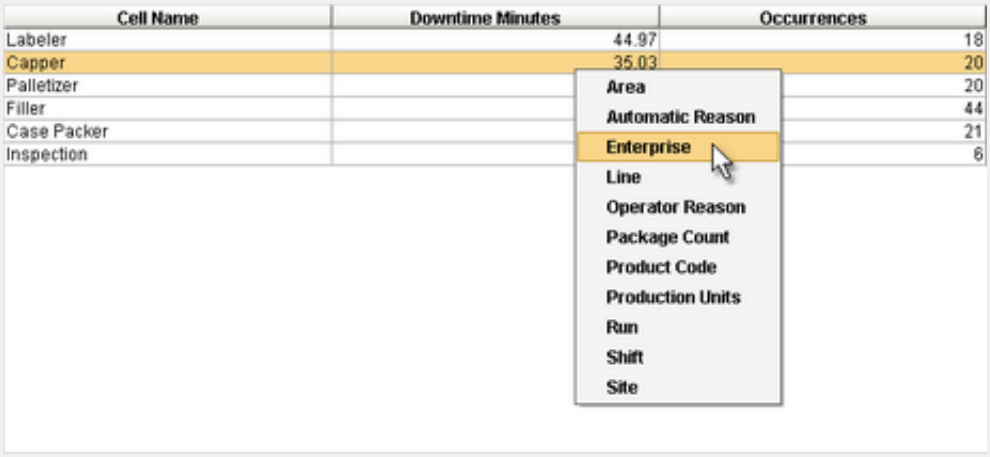
Customizers

Table Customizer shown below manages the data entered into the Analysis Table.



Examples

When the user clicks on a row in the table, the drill down menu will appear. When an item in the drill down menu is clicked on, the drillDown event is fired. Script in the drillDown event is responsible for updating the Data property to change the results shown in the table. The drill down menu information is set through the Drill Down Options property. The Drill Down Options can be populated from the Analysis Controller, Analysis Selector, SQL Query, scripting, or it can be manually defined in the designer.

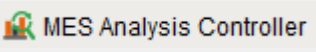


Component Analysis Table

Analysis Table

MES Analysis Controller

General

Component Palette Icon: 

Description

The analysis controller is an invisible component that makes analysis data available for reports and other components. The term invisible component means that the controller component appears in the designer, but is not visible from the client.

Properties

Name	Scripting	Category	Property Type	Description
Refresh on Settings Change	refreshOnSettingsChange	Data	Boolean	If true, automatically refresh when analysis settings values change
Data	data	Data	DataSet	Analysis results.
Ignition Dataset		Data	Dataset	Analysis results returned in an Ignition component friendly dataset. This property is not visible from the properties tab, but is accessbile for bindings to other components.

Analysis Settings Source	analysisSettingsSource	Data	int	The source of the analysis settings. If set to **Component**, the component Data Points, Filter By, etc. property settings are used. If set to **Saved**, then the named analysis settings are used.
Analysis Settings Name	analysisSettingsName	Data	String	Name of the analysis settings.
Filter By	filterBy	Data	String	The filter section allows you to limit the data that is included in the analysis. See Filter By section below to view the list.
Compare By	compareBy	Data	String	Compare Bys are the factors used to compare the analysis data. See Compare By section below to view the list.
Order By	orderBy	Data	String	Order Bys are the factors used to sort the analysis data.
Data Points	dataPoints	Data	String	Data points are the individual pieces of information that will be present in the analysis.

Name	Scripting	Category	Property Type	Description
				See Data points section below to view the list.
Start Date	startDate	Data	DateTime	Start Date.
End Date	endDate	Data	DateTime	End Date.
Error Message	errorMessage	Data	String	Error Message.
Execution Time (ms)	executionTime	Data	long	Analysis execution time in milliseconds.

Extension Functions

getParameterValue

- Description

Called to get a parameter value.

- Parameters

self - A reference to the component that is invoking this function

name - The parameter name as a string.

- Return

The parameter value.

- Scope

Client

Code Snippet

```
def getParameterValue(self, name):
```

```
    ##In this case, we check for the parameter named "eqPath",
short for Equipment Path.
    ##Return the parameter value as the Equipment Item Path of
the MES Object Selector in the root container.
    ##Note that this enables the use of the Stored Analysis in
a Report (with parameter "@eqPath") and in vision module
    ##screens with this Extension Function.
    if name == 'eqPath':
        return self.parent.getComponent('MES Object Selector').
equipmentItemPath
```

beforeUpdate

- Description

Called just before analysis data is refreshed.

- Parameters

self - A reference to the component that is invoking this function

- Return

Nothing

- Scope

Client

afterUpdate

- Description

Called just after analysis data is refreshed.

- Parameters

self - A reference to the component that is invoking this function.

data - The dataset that contains the new analysis data.

- Return

Nothing

- Scope

Client

Custom Analysis Settings

Right click on the MES analysis Controller and select **Custom Analysis Settings**. The MES Analysis Selector component is displayed that allows you to select a stored analysis or create new analysis settings for this analysis controller

Creating an Analysis Setting

Step 1: Click on the  icon

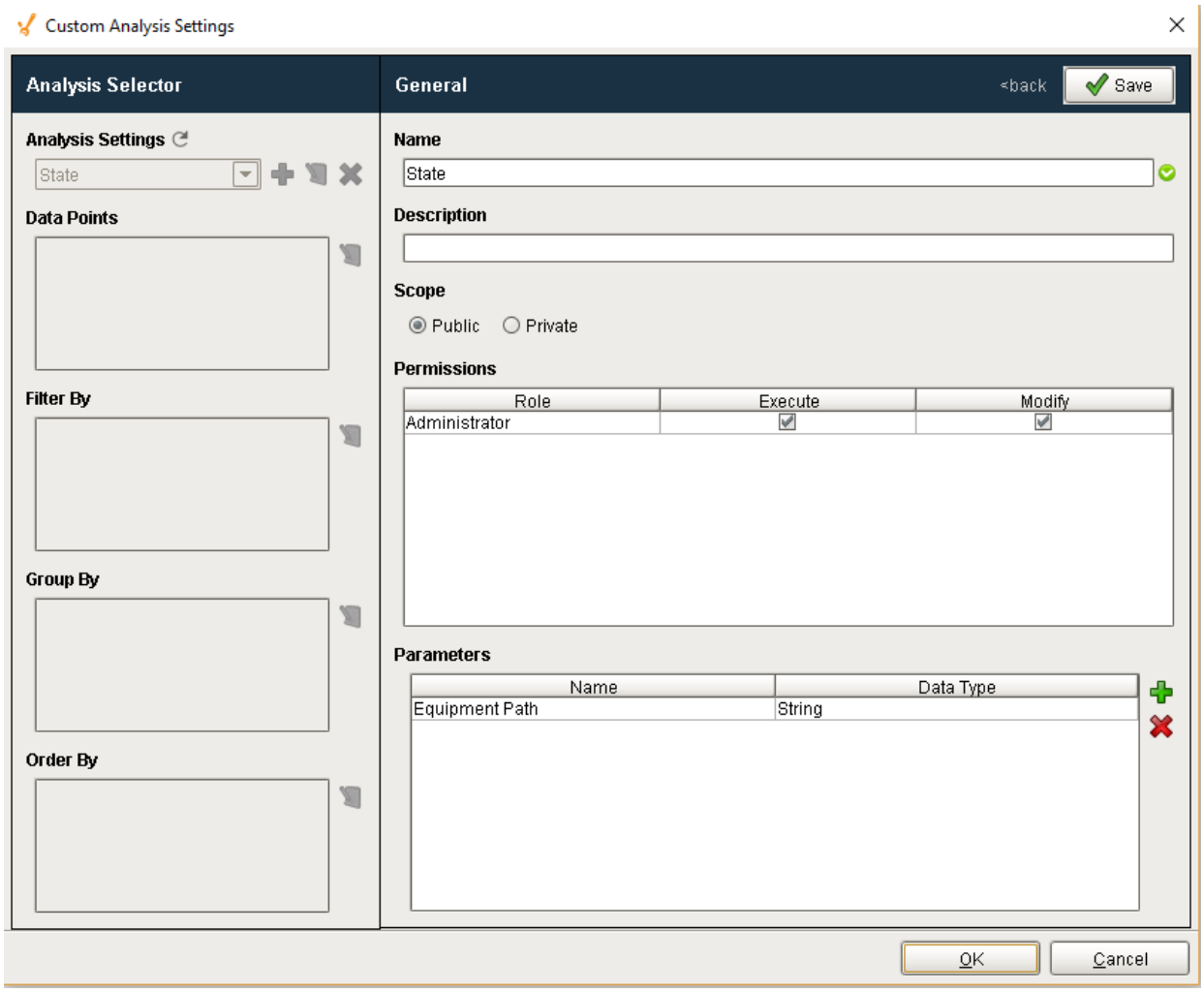
Step 2: Give the setting a name.

Step 3: Set the permissions of who can execute or modify these settings.

Step 4: Add parameters.

Step 5: Save the setting.

For more details on adding **Data Points**, **Filter By**, **Group By** and **Order By**, please refer to the MES Analysis Selector.



Examples

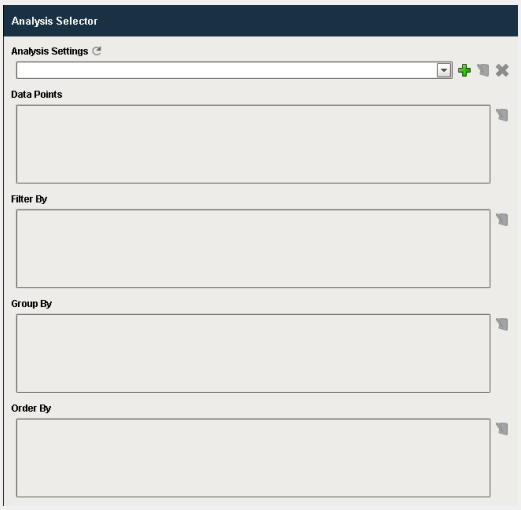
Here's an example to add columns into the dataset of the analysis controller from a custom property dataset.

Code Snippet

```
#Get the table data from the Analysis Controller
ds1 = event.source.parent.getComponent('Analysis Controller').
tableData
colCount = ds1.getColumnCount()

#Get the custom property 'Area'
columnName = event.source.parent.getComponent('Analysis
Controller').Area #Area is the column to be added
columnData = []
for i in range(ds1.getRowCount()):
     columnData.append(i* 10)
#Adds 'Area' to the Analysis Controller's dataset
ds2 = system.dataset.addColumn(ds1, colCount, columnData,
columnName, int)
event.source.parent.getComponent('Analysis Controller').
tableData = ds2
```

MES Analysis Selector

General



Component Palette Icon: MES Analysis Selector

Description

The MES analysis selector component allows for ad hoc selection of analysis data.

Properties

Name	Scripting	Category	Property Type	Description
Refresh on Settings Change	refreshOnSettingsChange	Data	Boolean	If true, automatically refresh when property values change.
Data	data	Data	Dataset	The dataset that contain the analysis results.
Ignition Dataset	ignitionDataset	Data	Dataset	Analysis results returned in an Ignition component friendly dataset. This property is not visible from the properties tab, but is accessbile for bindings to other components.
Drill Down Options	drillDownOptions	Data	Dataset	Dataset containing drill down options.
	previousDrillDownEnabled	Data	Boolean	

Name	Scripting	Category	Property Type	Description
Previous Drill Down Enabled				If true, then prevDrillDown.
Message	message	Data	String	Message returned with the analysis results.
Drill Down Bread Crumb	drillDownBreadCrumb	Data	String	A string representing the drill down path.
showMessage	Show Message	Data	Boolean	If true, show the message returned with the analysis results.
Start Date	startDate	Data	DateTime	The start date to get filter values.
End Date	endDate	Data	DateTime	The end date to get filter values.
Execution Time (ms)	executionTime	Data	long	Analysis execution time in milliseconds.
Title Font	titleFont	Appearance	Font	The font of text of the title bar.
	titleForeground	Appearance	Color	

Name	Scripting	Category	Property Type	Description
Title Foreground Color				The foreground color of the title bar.
Title Background Color	titleBackground	Appearance	Color	The background color of the title bar.
slidePanelWidth	Slide Panel Width	Appearance	int	The width of the slide panel.

Scripting

Scripting Functions

clearDrillDownHistory

- Description

Removes the drill down history .

- Parameters

None

- Return

Nothing

- Scope

Client

drillDown

- Description

Sets all the analysis selections to new state dictated by the drill down definition.

- Parameters

String compareByName - The compareBy definition to base the drill down .

String filtervalue - The value for filtering the analysis selections.

- Return

Nothing

- Scope

Client

prevDrillDown

- Description

Sets all the analysis selections to the previous state before the last drill down.

- Parameters

None

- Return

Nothing

- Scope

Client

Extension Functions

getParameterValue

- Description

Called to get a parameter value.

- Parameters

self - A reference to the component that is invoking this function

name - The parameter name as a string.

- Return

The parameter value.

- Scope

Client

Code Snippet

```
def getParameterValue(self, name):
    ##In this case, we check for the parameter named
"eqPath", short for Equipment Path.
    ##Return the parameter value as the Equipment Item Path
of the MES Object Selector in the root container.
    ##Note that this enables the use of the Stored Analysis
in a Report (with parameter "@eqPath") and in vision module
    ##screens with this Extension Function.
    if name == 'eqPath':
        return self.parent.getComponent('MES Object
Selector').equipmentItemPath
```

beforeUpdate

- Description

Called just before analysis data is refreshed.

- Parameters

self - A reference to the component that is invoking this function

- Return

Nothing

- Scope

Client

afterUpdate

- Description

Called just after analysis data is refreshed.

- Parameters

self - A reference to the component that is invoking this function.

data - The dataset that contains the new analysis data.

- Return

Nothing

- Scope

Client

Event Handlers

propertyChange

propertyChange

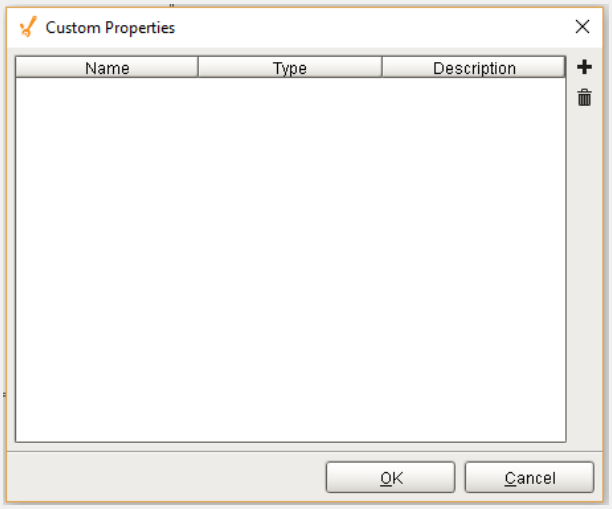
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

Custom Properties

The custom properties can be used to add user defined properties.



Filters

A filter can be added by selecting the  link to the right of **Filter By**. The following window panel will open and filter categories will be displayed. Multiple filters can be added using expressions. To add expression first select a filter and then a logical operator (horizontal list). Now select a operator (vertical list) and then add the next filter.

Filters follow a format similar to SQL in that you can use AND, OR and LIKE operators. You can also pass parameters to the filter expressions. Use '*' as a wildcard in your expressions.

Using Parameters

Both the Analysis Selector and Analysis Controller support parameter passing to filters in two different ways.

1. You can use the getParameterValue() extension function to pass the value to the named parameter
2. You can add a custom property to the component with the same name as the filter parameter and populate it with the value you want to use for the filter

You can't use both methods at the same time. If you use the custom property method, do not use the getParameterValue() extension function at the same time.

Parameters that are added as custom properties will appear in the **Parameter** dropdown box. They will not appear if added to the getParameterValue() extension function but can be typed in.

Filter Example

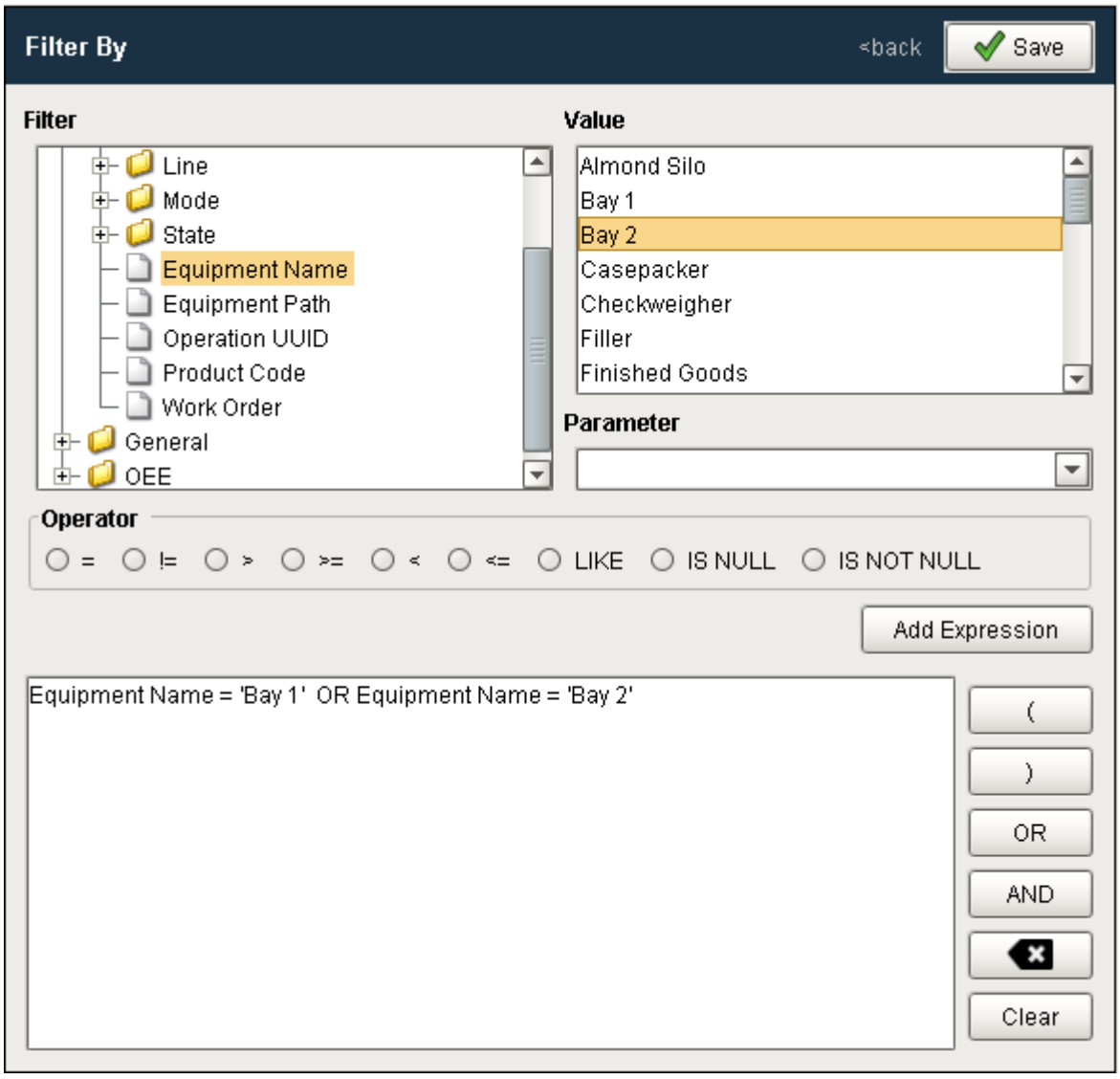
Operation UUID != '' AND Equipment Path = @LinePath AND Shift LIKE @ShiftName

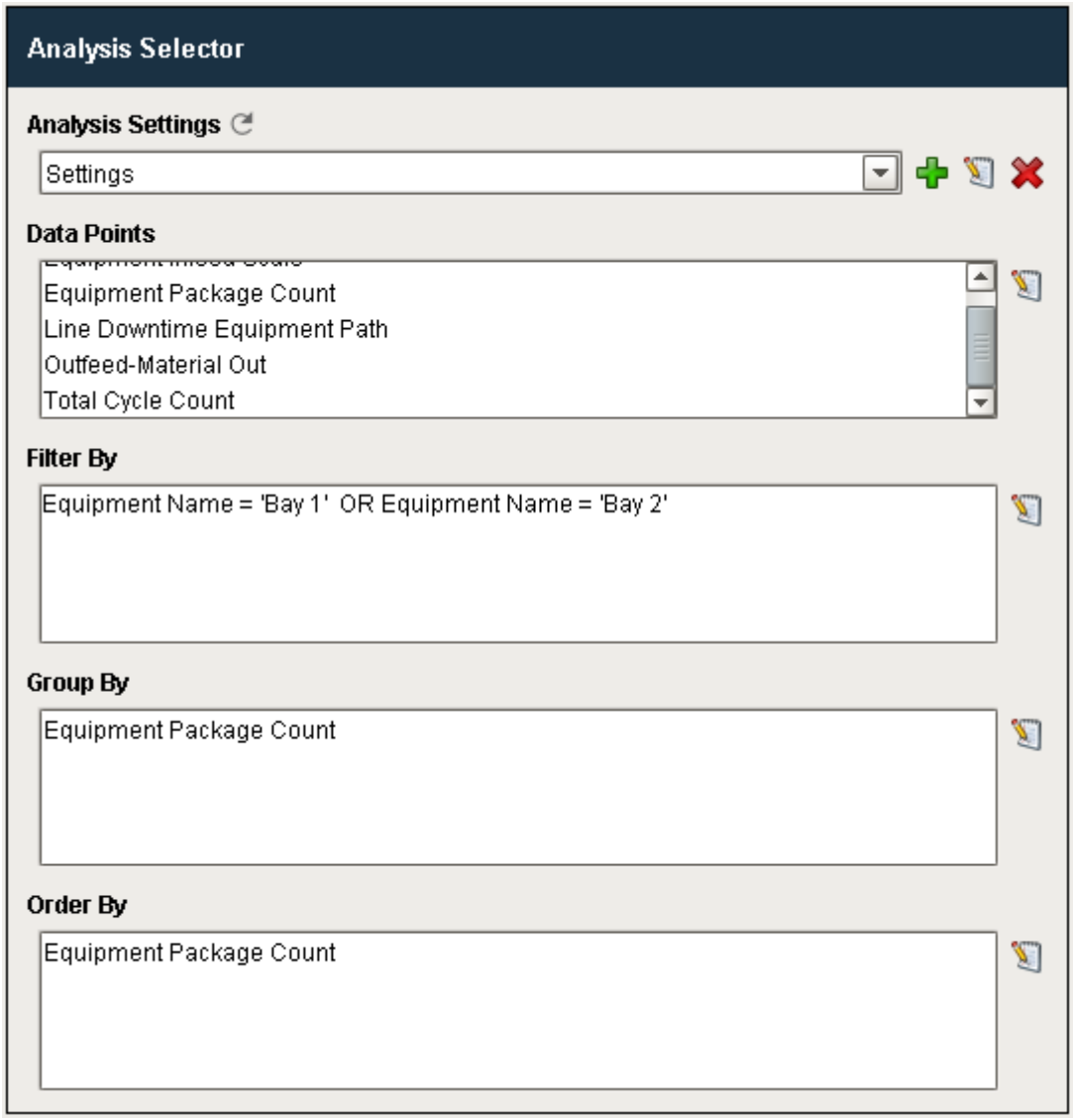
Click the  link by the filter category and specific filter items will be displayed.

When selected they will be added to the filters as shown below.

To minimize the number of filter options, only the options for the selected date range defined by the Start Date and End Date properties will be shown.

Compare By and **Data Points** work similarly to **Filter By** except there are no categories for these selections, just items. Selections can be removed by clicking  icon and unchecking the appropriate box. Analysis settings can be deleted by hitting the  delete icon.


Filter By
<back
Save
Filter
Line
Mode
State
Equipment Name
Equipment Path
Operation UUID
Product Code
Work Order
General
OEE
Value
Almond Silo
Bay 1
Bay 2
Casepacker
Checkweigher
Filler
Finished Goods
Parameter
Operator
= != > >= < <= LIKE IS NULL IS NOT NULL
Add Expression
Equipment Name = 'Bay 1' OR Equipment Name = 'Bay 2'
(
)
OR
AND
Clear



Data Points and Settings

Analysis Data Points and Settings are used by Live Analysis, the MES Analysis Selector and MES Analysis Controller components, the MES Analysis Data Source for reporting and the MES Analysis Settings object.

Equipment Data Points

Data Point	Data Type	Description
Equipment		

Data Point	Data Type	Description
Equipment Cell Order	Int4	Integer value that determines the cell order of the equipment within the line. Is set to *null* for the line. Is set to 0 for first cell within each cell group
Equipment Name	String	Name of the equipment as defined in the production model
Equipment Note	String	Any note that has been recorded for this piece of equipment through the Note tag collector path in the Production model will be exposed here.
Equipment Operation Begin	DateTime	Start Date time of the currently running operation on this equipment
Equipment Operation Sequence	Int4	The ordinal number (integer) of operation
Equipment Path	String	Production model path for this equipment
Equipment Type	String	Can be Line, Cell Group or Cell
Execution Time (ms)	Int8	Time taken to execute and update the Live Analysis. Used mainly for performance debugging
From Time Stamp	DateTime	Start Date Time of current data point results
Infeed Units	String	See Infeed Units for more details
Is Key Cell	Boolean	See Key Reason for more details
	String	Unique Identifier for currently running operation

Data Point	Data Type	Description
Operation UUID		
Outfeed Units	String	See Outfeed Units for more details
Product Code	String	Product code currently being processed on this equipment
Rate Period	String	See Rate Period for more details
Reject Units	String	See Reject Units for more details
To Time Stamp	DateTime	End Date Time of current data point results
Work Order	String	Work order currently being processed on this equipment

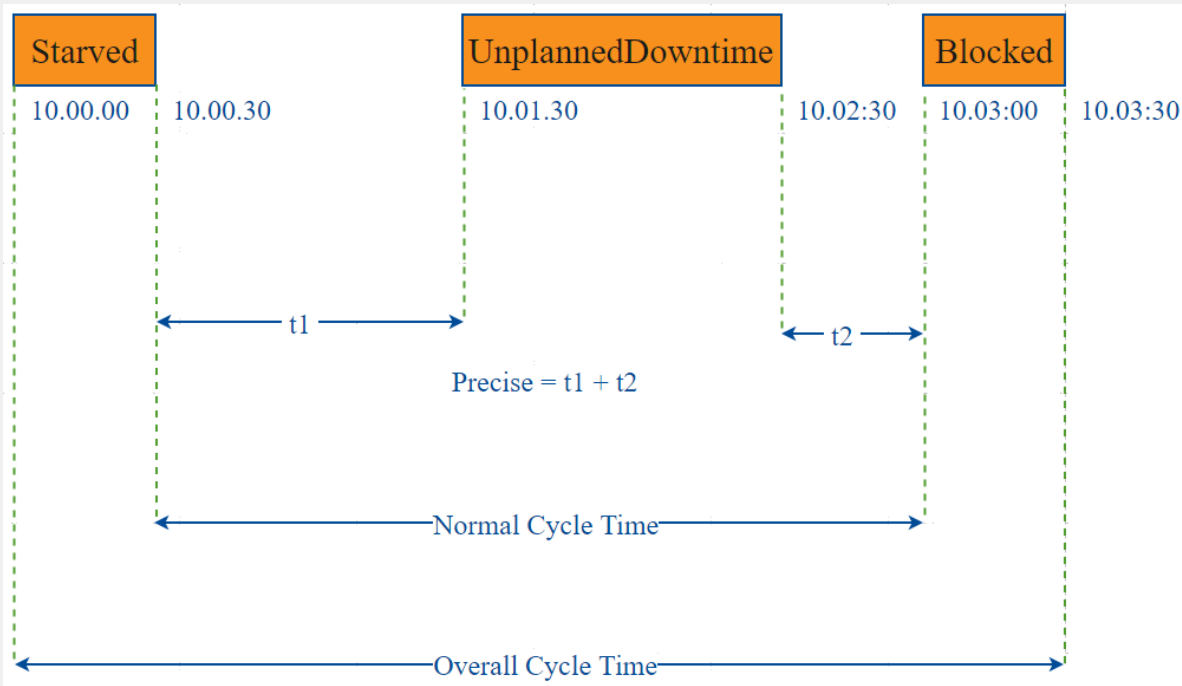
Equipment Count Data Points

Data Point	Data Type	Description
Equipment\Count		*Any defined counters for the production item will also appear in this folder
Equipment Infeed Scale	Float8	See Infeed Count Scale for more details
Equipment Package Count	Float8	See Package Count for more details
Equipment Reject Scale	Float8	See Reject Count Scale for more details
	String	

Data Point	Data Type	Description
Outfeed-Material Out		Value of the default MES Counter used for OEE outfeed count

Equipment Cycle Time Data Points

The Cycle Time data points provides a number of metrics that can be used to measure the amount of time required to produce one piece. It is often used to gain an understanding of variations in production. Live Analysis provides Target, Normal, Overall and Precise Cycle Time metrics.



Data Point	Data Type	Description
Equipment\Cycle Time		
Relative Cycle Count	String	Relative Cycle Count is how many occurred for the compare by.
	Float8	

Data Point	Data Type	Description
Target Cycle Time		Also known as Takt time, it is how often a piece must be produced to meet customer demand. It is often used to pace a production line, and it is a calculated number in seconds.
Total Cycle Count	String	Total Cycle Count is accumulative, it is sum total of all the cycle count.
Equipment\Cycle Time\Normal		**Normal Cycle Time is the actual cycle ignoring the equipment states like starved, blocked, etc.**
Average Normal Cycle Time	Float8	Average Normal cycle time in seconds for the time period selected
Max Normal Cycle Time	Float8	Max Normal cycle time in seconds for the time period selected
Min Normal Cycle Time	Float8	Min Normal cycle time in seconds for the time period selected
Normal Cycle Time	Float8	Normal Cycle Time in seconds is the actual cycle ignoring the equipment states like starved, blocked, etc.
Equipment\Cycle Time\Overall		**Overall Cycle Time is the cycle including states like downtime, starved, blocked, etc.**
Average Overall Cycle Time	Float8	Average Overall cycle time in seconds for the time period selected
Max Overall Cycle Time	Float8	Max Overall cycle time in seconds for the time period selected
Min Overall Cycle Time	Float8	Min Overall cycle time in seconds for the time period selected
	Float8	

Data Point	Data Type	Description
Overall Cycle Time		Overall Cycle Time in seconds is the cycle including states like downtime, starved, blocked, etc.
Equipment\Cycle Time\Precise		**Precise Cycle Time is the cycle time ignoring all the equipment states**
Average Precise Cycle Time	Float8	Average Precise cycle time in seconds for the time period selected
Max Precise Cycle Time	Float8	Max Precise cycle time in seconds for the time period selected
Min Precise Cycle Time	Float8	Min Precise cycle time in seconds for the time period selected
Precise Cycle Time	Float8	Precise cycle time in seconds excluding states like planned downtime, unplanned downtime, starved and blocked.

Line Data Points

The Line Data points returns data for the line regardless of the Equipment the Live Analysis has been set up for.

Data Point	Data Type	Description
Line /Downtime		
Line Downtime Equipment Name	String	Name of the equipment that is responsible for causing line downtime
	String	Production model equipment path for equipment that is responsible for causing line downtime

Data Point	Data Type	Description
Line Downtime Equipment Path		
Line Downtime Event Sequence	Int4	Every downtime event on the line is provided with an incrementing sequence number
Line Downtime Note	String	Note entered at the Line level
Line Downtime Occurrence Count	Int4	Number of downtime events for the selected period.
Line Downtime Reason	String	The line or cell group (sub line) downtime reason. 1. When the line is down the Line Downtime Reason is the same as the Line State Name. 2. When the line is up the Line Downtime Reason is blank.
Line Downtime Reason Path	String	The full reason name for line or cell group (sub line) downtime reason. Line State name including State Class i. e. *Default/Cell Faulted*
Line Downtime Reason Split	Boolean	The line downtime reason split indicator. True is current downtime event has been split into multiple downtime events

Data Point	Data Type	Description
Line Downtime State Time Stamp	DateTime	The time stamp for the equipment state change of the cell group (sub line) or cell that caused the line down time even.
Line /Meantime		
Line MTBF	Float8	The calculated Meantime (minutes) Between Failure for the selected period. Refer to Setting Up Equipment States - Meantime Metrics for more details.
Line Meantime Metrics Enabled	Boolean	Returns if Meantime metrics have been enabled for this equipment.
Line/Schedule		
Line Schedule Available	Boolean	True if this operation was scheduled
Line Schedule Available Time	Float8	Time in minutes for available production time adjusted for line schedule availability and mode.
Line Standard Count	String	Amount of product that should have been produced based on the line schedule available time and line standard rate
Line Standard Count Variance	String	Variance between standard count and actual count
Line Target Count	String	Amount of product that should have been produced based on the line schedule available time and line schedule rate
	String	

Data Point	Data Type	Description
Line Target Count Variance		Variance between line scheduled count and line OEE outfeed count.
Schedule Rate	Float8	See Schedule Rate for more details
Line/State		
Line State Duration	Float8	The line or cell group (sub line) downtime event duration in minutes.
Line State Event Begin	DateTime	The line or cell group (sub line) downtime event begin date time.
Line State Event End	DateTime	The line or cell group (sub line) downtime event end date time.
Line State Event Sequence	Int4	The equipment state event sequence number.
Line State Name	String	The line or cell group (sub line) state. 1. When the line is down the Line Downtime Reason is the same as the state name. 2. When the line is up the Line Downtime Reason is blank.
Line State Override Scope	String	The state override scope for a line or cell group (sub line). See Setting Up Equipment - Override Scope for more details
Line State Override Type	String	The state override type for a line or cell group (sub line). See Setting Up Equipment - Override for more details

Data Point	Data Type	Description
Line State Type	String	The line or cell group (sub line) state type. See Setting Up Equipment - State Type for more details
Line State Value	Int4	The line or cell group (sub line) downtime state code. See Setting Up Equipment - State Code for more details

Equipment Mode & State Data Points

Data Point	Data Type	Description
Equipment /Mode		
Equipment Mode Name	String	Name of the current mode. See Setting Up Equipment Modes for more details
Equipment Mode Type	String	Name of the current mode type. See Setting Up Equipment Modes for more details
Equipment Mode Value	Int4	Name of the current mode. See Setting Up Equipment Modes for more details
Mode Begin Time	DateTime	Start time of the current mode
Mode Duration	Float8	Duration of the current mode in minutes
Mode End Time	DateTime	End time of the current mode
OEE Enabled	Boolean	See Setting Up Equipment Modes - OEE Enabled for more details
	Boolean	See Setting Up Equipment Modes - OEE Enabled for more details

Data Point	Data Type	Description
Production Counts Enabled		
Equipment /State		
Equipment Original State Value	Int4	The original value of equipment state tag collector before it is updated by using MES Value Editor component or scripting
Equipment State Name	String	Current state name
Equipment State Path	String	Production model equipment path for equipment that is responsible for causing line downtime
Equipment State Split	Boolean	True is current downtime event has been split into multiple downtime events
Equipment State Type	String	See Setting Up Equipment - State Type for more details
State Begin Time	DateTime	Start time of the current state
State Duration	Float8	Duration of current state in minutes
State End Time	DateTime	End time of the current state

Equipment Meantime Data Points

Data Point	Data Types	Description
Equipment/Meantime		
Equipment MTBF	Float8	The Mean Time (minutes) Between Failure for the selected period
Equipment Meantime Metrics Enabled	Boolean	True if Equipment Meantime Metrics are enabled for the current equipment state

Equipment General Data Points

Data Point	Data Types	Description
Equipment /General		
Delta Time Stamp	Float8	Time gap (minutes) between the rows of data.
From Time Stamp	DateTime	Start time (minutes) of the current period
Shift	String	Name of the current shift as set by the Ignition Schedule Management component and defined for the current line or by the value passed in the equipment shift tag collector
Shift Day Text	String	Name of the current day
Shift Day of Month	Int4	Int value of the current month
	Int4	Int value of the current day of the week

Data Point	Data Types	Description
Shift Day of Week		
Shift Day of Year	Int4	Int value of the current day of the year
Shift ISO Week of Year	Int4	Int value of the ISO week of the year
Shift Month Text	String	Name of the current month
Shift Month of Year	Int4	Int value of the current month of the year
Shift Start Date	DateTime	Start time of the current shift
Shift Week of Month	Int4	Int value of the current week of the month
Shift Week of Year	Int4	Int value of the current week of the year
Shift Year	Int4	Int value of the current year
To Time Stamp	DateTime	Endtime (minutes) of the current period

Equipment OEE Data Points

Data Point	Data Type	Description

Data Point	Data Type	Description
Equipment/OEE		
Elapsed Time	Float8	Elapsed Time of current operation
OEE	Float8	OEE value for selected period
OEE General Count	Long	Any count value other than infeed, outfeed, reject and waste value for the selected time period
OEE Infeed Count	Long	Equipment infeed count value for the selected period
OEE Infeed Count Equipment Path	String	Infeed count tag collector path
OEE Outfeed Count	Long	Equipment outfeed count value for the selected period
OEE Outfeed Count Equipment Path	String	Outfeed count tag collector path
OEE Reject Count	Long	Equipment reject count value for the selected period
Planned Downtime	Float8	Planned Downtime duration (Double) for selected period
Runtime	Float8	Planned Downtime duration (Double) for selected period
Short Stop Time	Float8	Short stop duration (Double) for selected period
Standard Rate	Float8	See standard rate for more details
Target Changeover Time	Float8	Amount of time in minutes set for Target Changeover. See Changeover Duration for more details
Unplanned Downtime	Float8	Unplanned Downtime duration (Double) for selected period

Data Point	Data Type	Description
Equipment/OEE /Availability		
Is Short Stop	Boolean	True if current equipment state is consider a shortstop. See Short Stop Threshold for more details
OEE Availability	Float8	OEE Availability value for selected period
Equipment/OEE /Performance		
OEE Performance	Float8	OEE Performance value for selected period
Infeed Standard Count	Float8	Calculated expected infeed based on standard rate
Equipment/OEE /Quality		
OEE Quality	Float8	OEE Quality value for selected period

Setting Values

The analysis results that are returned can be modified through the use of settings. Setting values provide a number of keywords as listed below.

Format for entering the keywords is k*eyword1=True, keyword2=100.0, keyword3=10.*

> ⚠ Settings like Enable Totalized Mode, Include Future, Last Values and Rollup Time span is meant for analysis selector and not for live analysis

Setting	Description	Use	Example
Date Format	Date format fields can be customized with this setting e.g. 'YYYY/MM/dd hh:mm:ss a'	All	

Setting	Description	Use	Example
			Date Format = `2017/04/12 19:45:30`
Enable Totalized Mode	This setting accumulates the count. Useful for charts where you wish to display the accumulated production count over time	Not valid for Live Analysis	Enable Totalized Mode = True
Include Future	Allows for count values to be calculated in the future. Useful for charts where you want to display target counts for future runs	Not valid for Live Analysis	Include Future = True
Last Values	Only the latest values are shown.	Not valid for Live Analysis	Last Values = True
OEE Availability Cap	The maximum value calculated can be capped with this setting	All	OEE Availability Cap = 100.0
OEE Performance Cap	The maximum value calculated can be capped with this setting	All	OEE Performance Cap = 100.0
OEE Quality Cap	The maximum value calculated can be capped with this setting	All	OEE Quality Cap = 100.0
Rollup Time Span	If the time (seconds) between downtime events is less than the rollup time and it is the same equipment and reason, then it will rollup the event into one row in the results and will increase the occurrence count.	Not valid for Live Analysis	Rollup Time Span = 30

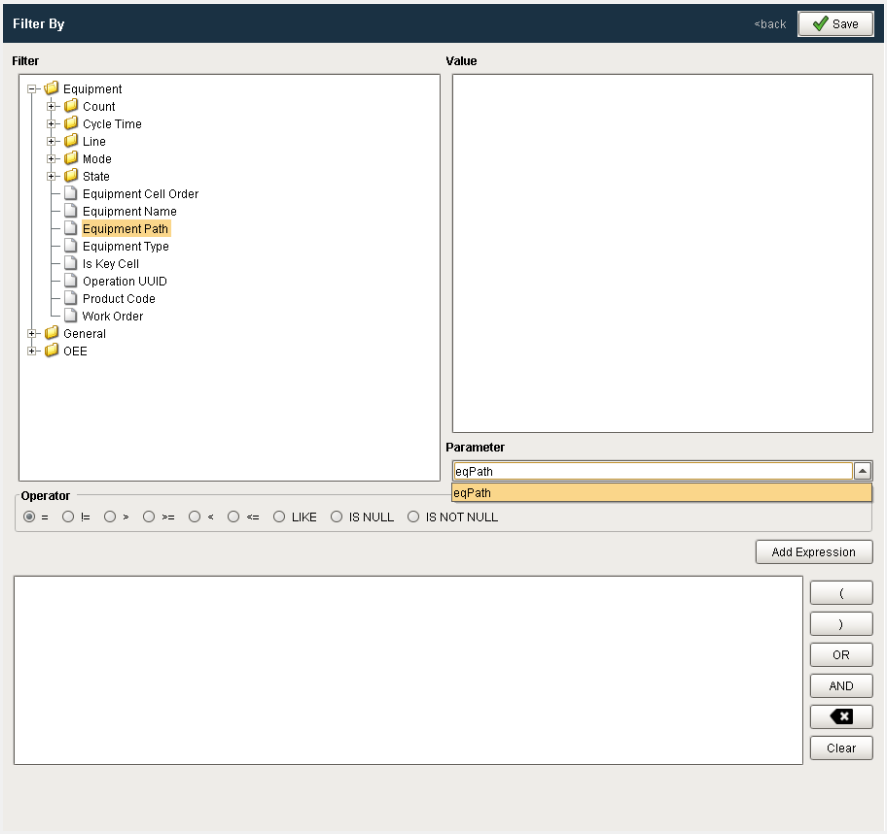
Setting	Description	Use	Example
Row Limit	The analysis can be limited to a certain number of rows.	All	Row Limit = 10

Reporting Example

The Analysis Selector is exposed to support the configuration of "MES Analysis" data sources in the Ignition Reporting Module Data tab. Stored Analysis settings used on the operator screens may also be applied to reports.

A difference for Reports over Screens is the use of Report Parameters to bind data automatically.

In this example the Equipment Path property will be bound to the Report parameter **eqPath** . This will allow an equipment path to be passed to the report and used in the analysis rather than statically assigning the path.

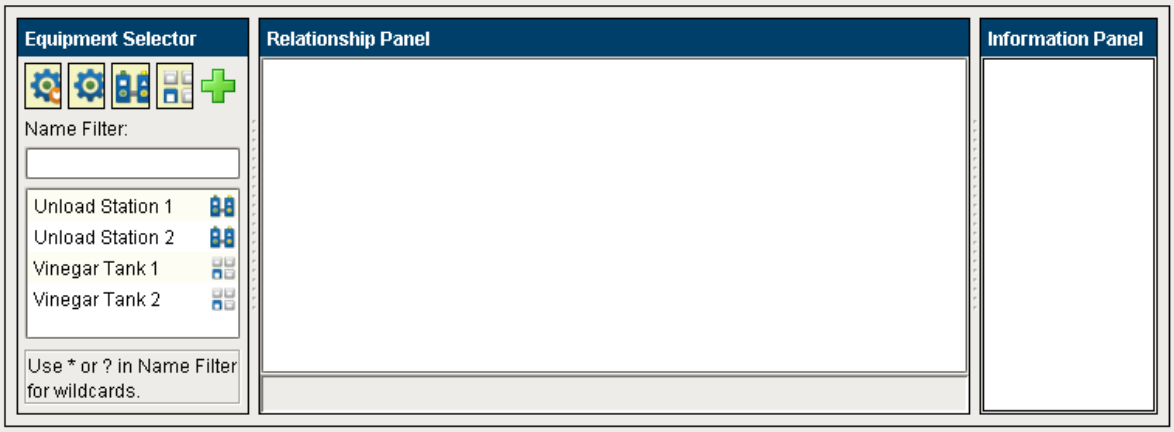


When selected with the **Add Expression** button, the filter will be shown as "Equipment Path = @eqPath". Since this notation differs slightly from custom properties in the vision module, check the above example for the extension function getParameterValue() which will set the parameter value so that the same Stored Analysis settings may be used for Reports and On-Screen analysis.

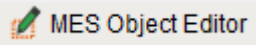
The rest of the features function like the Vision Module version of the Analysis Selector.

MES Object Editor

General



Component Palette Icon:



Description

MES Object Editor is a component to manage MES objects. In addition to using this component to manager MES objects, script can also be used.

The MES Object Editor component is used to edit resources, segments and operations. See ISA-95 for more information about the various MES objects. Equipment can be put into categories by first adding a new equipment category and then adding equipment to it. The same can be done for material and personnel.

Properties

Name	Scripting	Category	Property Type	Desc
Read Only	readOnly	Behavior	Boolean	If tru editi allow
User Menu Items	userMenuItems	Behavior	DataSet	A da store men
Editor Mode	editorModeValue	Data	int	Set v MES the e displ editi
Enable Auto Sizing	enableAutoSizing	Behavior	Boolean	If tru size mou on o spac
Menu Add New Icon Path	menuAddNewIconPath	Appearance	String	The path icon appe the a men
Menu Edit Icon Path	menuEditIconPath	Appearance	String	The path icon appe the r men
Menu Delete Icon Path	menuDeleteIconPath	Appearance	String	

Name	Scripting	Category	Property Type	Desc
				The path icon appe the c men
Menu Show Refrences Icon Path	menuShowReferencesIconPath	Appearance	String	The path icon appe the s refer men
Menu Stop Show Refrences Icon Path	menuStopShowReferencesIconPath	Appearance	String	The path icon appe the s refer men
Enable Show Deleted Properties	enableShowDeleted	Appearance	Boolean	Allov delet prop be sl
Show Deleted Objects	showingDeletedObjects	Appearance	Boolean	Dele obje show
Node Configuration	nodeConfiguration	Behavior	DataSet	A da store confi
Tab Font	tabFont	Appearance	Font	

Name	Scripting	Category	Property Type	Desc
				Font of ec pane
Editor Title Font	editorTitleFont	Appearance	Font	Font of ec pane
Category Font	categoryFont	Appearance	Font	Font cate in ec table
Property Font	propertyFont	Appearance	Font	Font prop editi
Description Area Font	descriptionAreaFont	Appearance	Font	Font desc area editi
Button Font	buttonFont	Appearance	Font	Font butto
Close Button Font	closeButtonFont	Appearance	Font	The the b close editi
Popup Options Font	popupOptionsFont	Appearance	Font	Font popu optic
Popup Message Font	popupMessageFont	Appearance	Font	Font popu mess

Name	Scripting	Category	Property Type	Desc
Miscellaneous Font	miscellaneousFont	Appearance	Font	Font misc comp
Title Background Color	titleBackgroundColor	Appearance	Color	The the t back the e pane
Title Text Color	titleTextColor	Appearance	Color	The the t the e pane
Close Button Color	closeButtonColor	Appearance	Color	The the b close editi
Category Background Color	categoryBackgroundColor	Appearance	Color	The back colo cate rows
Edge Color	edgeColor	Appearance	Color	The the li betw node
Primary MES Object Filter	primaryMESObjectLink	Data	MESObjectLink	The MES to sh edito
Mode	mode	Data	String	

Name	Scripting	Category	Property Type	Desc
				The mod edito

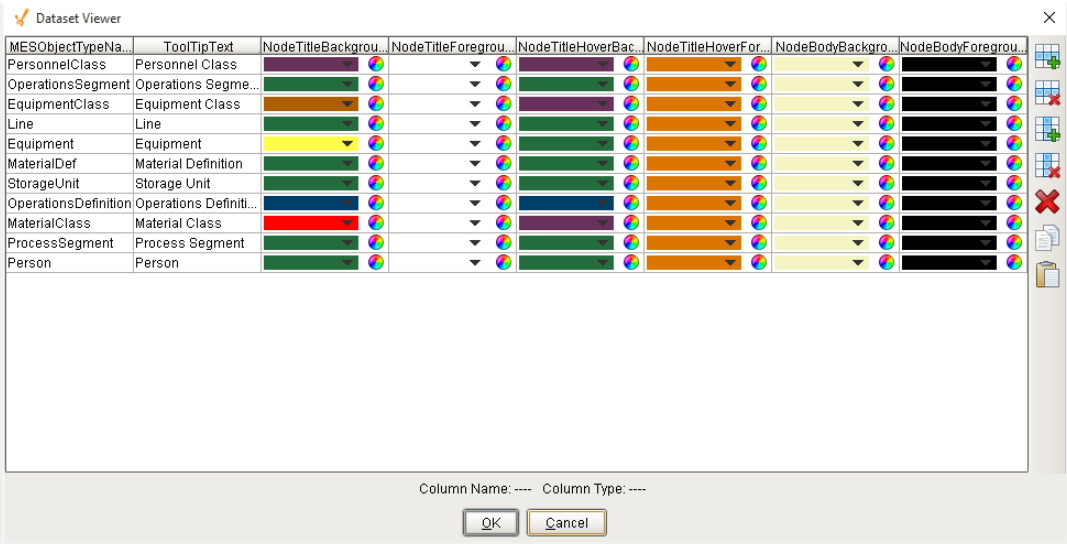
Node Configuration Property

A dataset containing node configuration used when displaying nodes in the MES Object Editors component. The colors of the different parts of a node can be changed using this property. This provides a visual deference of the different types of MES objects.

This property is a dataset allowing any number of node configurations keyed by first the name of the node and if a match is not found, then it will look for match by node type. If no matches are found, the entry named Default will be used.

It controls the appearance of each node. Click on the **Dataset Viewer** icon for the **Behaviour** property in **Property Editor** to set the values.

MESObjectTypeName is the name of the type of MES object that should be included as nodes. Default, MaterialClass, MaterialDef, ProcessSegment, EquipmentClass, Equipment, MES*, OperationsDefinition, OperationSegment, PersonnelClass, Person are the values inbuilt on Ignition, as shown below. ToolTipText provides a hint to visual components as to what should be displayed when the user hovers their mouse cursor over the component. NodeTitleBackgroundColor is the color of the node header. NodeTitleForegroundColor is the color of the text in the node header. NodeTitleHoverBackgroundColor is the color of the node header when your mouse hovers over it. The color of the text of the node header when user hover their mouser over it is controlled by NodeTitleHoverForegroundColor. Color of the node is determined by NodeBodyBackground. NodeBodyForeground is the color of the text inside the main body of node.



Scripting

Scripting Functions

autoFit

- Description

Zooms a display such that all items within a given group will fit within the display bounds. By default, this achieved by clicking the right mouse button once, with no dragging.

- Parameters

None

- Return

Nothing

- Scope

Client

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

Event Handlers

menu

userMenuItemClicked

This event fires when the menu item is clicked, or if the user selects the menu item using the keyboard and presses the Enter key. It can also occur if an access key or shortcut key is pressed that is associated with the MenuItem.

Property	Description
source	The component that fired this event.
menuItemName	Name of the user menu item that triggered the event.
nodeName	Name of the node. This is the same as the name of the MES object that is associated with the node.
objectType	Name of the MES object type that is associated with the node.
uuid	UUID of the MES object that is associated to the node.
lotUUID	UUID of the material lot.
lotName	Name of the material lot.
lotSequence	The sequence number associated with the material lot.
lotUse	The lot use type of the material.
beginDateTime	Date and Time at which the event was triggered.

Property	Description
materialUUID	UUID of the material.
materialName	Name of the material.
lotEquipmentUUID	UUID of the equipment lot.
lotEquipmentName	Name of the equipment lot.
segmentUUID	UUID of the segment.
segmentName	Name of the segment.
segmentEquipmentUUID	UUID of the segment equipment.
segmentEquipmentName	Name of the segment equipment.

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

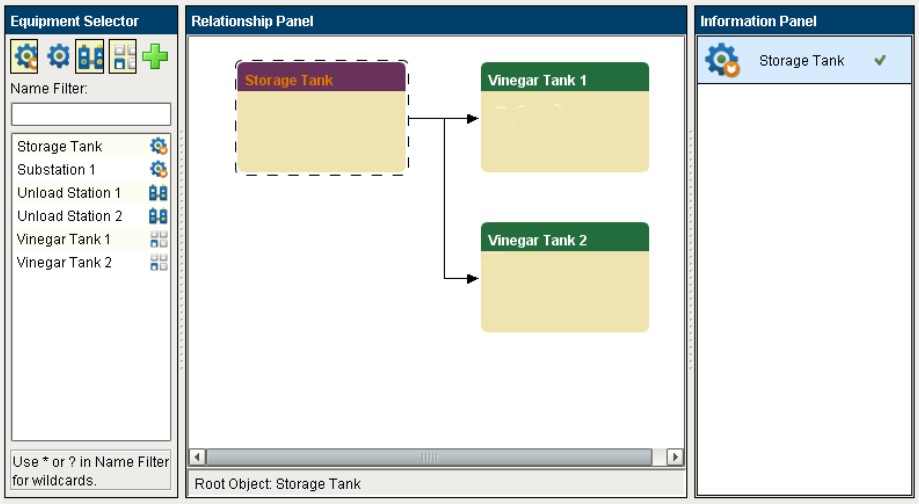
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

This component does not have any custom properties.

Examples



Property Name	Value
Editor Mode	Equipment
Left Split Pane Width	250
Right Split Pane Width	217
Node Configuration	

MESObjectTypeName	ToolTipText	NodeTitleBackgroundColor
PersonnelClass	Personnel Class	
OperationsSegment	Operations Segment	
EquipmentClass	Equipment Class	
Line	Line	
Equipment	Equipment	
MaterialDef	Material Definition	
StorageUnit	Storage Unit	
OperationsDefinition	Operations Definition	
MaterialClass	Material Class	
ProcessSegment	Process Segment	
Person	Person	

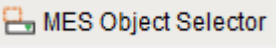
Info

For the MES Object Editor component to find the MES Object Selector, it must be in the same container on the window. It is okay to be in a container, they just both have to be in the same container or root container.

When a new class, segment or operation is added, the MES Object Selector selection will change to reflect the newly added MES object. This doesn't happen when adding a new child in order to keep the primary MES object shown.

MES Object Selector

General

Component Palette Icon: 

Description

A component to allow selection of MES objects. It contains many properties to filter the type and name of the MES object to include in the list.

Properties

Name	Scripting	Category	Property Type	Description
Custom Property Name Filter	customPropertyNameFilter	Data	String	Filter value, including * and ? wildcard characters, to filter

Name	Scripting	Category	Property Type	Description
				results by custom property names.
Custom Property Value Filter	customPropertyValueFilter	Data	String	List of custom property name and value to filter results by.
Equipment Item Path	equipmentItemPath	Data	String	The equipment path within the production model.
Include Equipment Class Objects	includeEquipmentClassObjects	Data	boolean	If true, includes MES equipment class objects.
Include Equipment Objects	includeEquipmentObjects	Data	boolean	If true, includes MES equipment class objects.
	includeMaterialClassObjects	Data	boolean	If true, includes MES

Name	Scripting	Category	Property Type	Description
Include Material Class Objects				material class objects.
Include Material Def Objects	includeMaterialDefObjects	Data	boolean	If true, includes MES equipment class objects.
Include MES Area Objects	includeMESAreaObjects	Data	boolean	If true, includes MES equipment class objects.
Include MES Enterprise Objects	includeMESEnterpriseObjects	Data	boolean	If true, includes MES equipment class objects.
Include MES Line Cell Group Objects	includeMESLineCellGroupObjects	Data	boolean	If true, includes MES equipment class objects.
Include MES Line Cell Objects	includeMESLineCellObjects	Data	boolean	

Name	Scripting	Category	Property Type	Description
				If true, includes MES equipment class objects.
Include MES Line Objects	includeMESLineObjects	Data	boolean	If true, includes MES material class objects.
Include MES Site Objects	includeMESSiteObjects	Data	boolean	If true, includes MES equipment class objects.
Include MES Storage Unit Objects	includeMESStorageUnitObjects	Data	boolean	If true, includes MES equipment class objects.
Include MES Storage Zone Objects	includeMESStorageZoneObjects	Data	boolean	If true, includes MES equipment class objects.
	includeOperationsDefinitionObjects	Data	boolean	

Name	Scripting	Category	Property Type	Description
Include Operations Definition Objects				If true, includes MES equipment class objects.
Include Operations Request Objects	includeOperationsRequestObjects	Data	boolean	If true, includes MES equipment class objects.
Include Operations Response Objects	includeOperationsResponseObjects	Data	boolean	If true, includes MES equipment class objects.
Include Operations Segment Objects	includeOperationsSegmentObjects	Data	boolean	If true, includes MES material class objects.
Include Person Objects	includePersonObjects	Data	boolean	If true, includes MES equipment class objects.
	includePersonnelClassObjects	Data	boolean	

Name	Scripting	Category	Property Type	Description
Include Personnel Class Objects				If true, includes MES equipment class objects.
Include Process Segment Objects	includeProcessSegmentObjects	Data	boolean	If true, includes MES equipment class objects.
Include Request Segment Objects	includeRequestSegmentObjects	Data	boolean	If true, includes MES equipment class objects.
Include Response Segment Objects	includeResponseSegmentObjects	Data	boolean	If true, includes MES equipment class objects.
Parent MES Object Filter	primaryClassFilter	Data	String	Parent MES object to filter the results by.
	includeOperationsSegmentObjects	Data	boolean	

Name	Scripting	Category	Property Type	Description
Parent MES Object Name Filter				The name of the parent MES object to filter the results by.
Parent MES Object Path	parentMESObjectPath	Data	String	The path of the parent MES object to filter the results by.
Selected MES Object	selectedMESObject	Data	String	The selected MES object link.
Selected Name	selectedName	Data	String	The selected MES object Name.
Selected UUID	selectedUUID	Data	String	The selected MES object UUID.

Scripting

Scripting Functions

clearSelection

- Description

Deselects the previous selection.

- Parameters

None

- Return

Nothing

- Scope

Client

getParentMESObjectFilter

- Description

Returns the parent MES object to filter the results by.

- Parameters

None

- Return

MESObjectLink parentMESObjectFilter - The parent MES object filter.

- Scope

Client

getParentMESObjectPath

- Description

Gets the path of the parent MES object to filter the results by.

- Parameters

None

- Return

String path - The path of the parent MES object.

- Scope

Client

getSelectedMESObject

- Description

Returns the selected MES object.

- Parameters

None

- Return

MESObjectLink objLink - The link to the selected MES object.

- Scope

Client

getSelectedMESObjectTypes

- Description

Returns the selected MES object types.

- Parameters

None

- Return

MESObjectTypes types - The object types of the selected MES objects.

- Scope

Client

setEquipmentItemPath

- Description

Sets the equipment item path.

- Parameters

String equipmentItemPath - The path to set for the equipment item.

- Return

Nothing

- Scope

Client

setExcludedEquipmentPath

- Description

Sets the path of the equipment to be excluded.

- Parameters

String excludedEquipmentPath - The path of the equipment to be excluded.

- Return

Nothing

- Scope

Client

setIncludeEquipmentClassObjects

- Description

If set to true, includes MES equipment class objects.

- Parameters

boolean includeEquipmentClassObjects - Set to true to include MES equipment class objects.

- Return

Nothing

- Scope

Client

setIncludeEquipmentObjects

- Description

If set to true, includes MES equipment objects.

- Parameters

boolean includeEquipmentObjects - Set to true to includes MES equipment objects.

- Return

Nothing

- Scope

Client

setIncludeMESAreaObjects

- Description

If set to true, includes MES area objects.

- Parameters

boolean includeAreaObjects - Set to true to includes MES area objects.

- Return

Nothing

- Scope

Client

setIncludeMESEnterpriseObjects

- Description

If set to true, includes MES enterprise objects.

- Parameters

boolean includeEnterpriseObjects - Set to true to includes MES enterprise objects.

- Return

Nothing

- Scope

Client

setIncludeMESLineCellGroupObjects

- Description

If set to true, includes MES line cell group objects.

- Parameters

boolean includeLineCellGroupObjects - Set to true to includes MES line cell group objects.

- Return

Nothing

- Scope

Client

setIncludeMESLineCellObjects

- Description

If set to true, includes MES line cell objects.

- Parameters

boolean includeLineCellObjects - Set to true to includes MES line cell objects

- Return

Nothing

- Scope

Client

setIncludeMESLineObjects

- Description

If set to true, includes MES line objects.

- Parameters

boolean includeLineObjects - Set to true to includes MES line objects

- Return

Nothing

- Scope

Client

setIncludeMESSiteObjects

- Description

If set to true, includes MES site objects.

- Parameters

boolean includeSiteObjects - Set to true to includes MES site objects

- Return

Nothing

- Scope

Client

setIncludeMESStorageUnitObjects

- Description

If set to true, includes MES storage unit objects.

- Parameters

boolean includeStorageUnitObjects - Set to true to includes MES storage unit objects

- Return

Nothing

- Scope

Client

setIncludeMESStorageZoneObjects

- Description

If set to true, includes MES storage zone objects.

- Parameters

boolean includeStorageZoneObjects - Set to true to includes MES storage zone objects.

- Return

Nothing

- Scope

Client

setIncludeMaterialClassObjects

- Description

If set to true, includes MES material class objects.

- Parameters

boolean includeMaterialClassObjects - Set to true to includes MES material class objects

- Return

Nothing

- Scope

Client

setIncludeMaterialDefObjects

- Description

If set to true, includes MES material definition objects.

- Parameters

boolean includeMaterialDefObjects - Set to true to includes MES material definition objects.

- Return

Nothing

- Scope

Client

setIncludeOperationsDefinitionObjects

- Description

If set to true, includes operations definition objects.

- Parameters

boolean includeOperationsDefinitionObjects - Set to true to includes MES operations definition objects

- Return

Nothing

- Scope

Client

setIncludeOperationsRequestObjects

- Description

If set to true, includes operations request objects.

- Parameters

boolean includeOperationsRequestObjects - Set to true to includes MES operations request objects

- Return

Nothing

- Scope

Client

setIncludeOperationsResponseObjects

- Description

If set to true, includes operations response objects.

- Parameters

boolean includeOperationsResponseObjects - Set to true to includes MES operations response objects

- Return

Nothing

- Scope

Client

setIncludeOperationsSegmentObjects

- Description

If set to true, includes operations segment objects.

- Parameters

boolean includeOperationsSegmentObjects - Set to true to includes MES operations segment objects.

- Return

Nothing

- Scope

Client

setIncludePersonObjects

- Description

If set to true, includes person objects.

- Parameters

boolean includePersonObjects - Set to true to includes MES person objects.

- Return

Nothing

- Scope

Client

setIncludePersonnelClassObjects

- Description

If set to true, includes personnel class objects.

- Parameters

boolean includePersonnelObjects - Set to true to includes MES personnel objects.

- Return

Nothing

- Scope

Client

setIncludeProcessSegmentObjects

- Description

If set to true, includes process segment objects.

- Parameters

boolean includeProcessSegmentObjects - Set to true to includes MES process segment objects.

- Return

Nothing

- Scope

Client

setIncludeRequestSegmentObjects

- Description

If set to true, includes request segment objects.

- Parameters

boolean includeRequestSegmentObjects - Set to true to includes MES request segment objects.

- Return

Nothing

- Scope

Client

setIncludeResponseSegmentObjects

- Description

If set to true, includes response segment objects.

- Parameters

boolean includeResponseSegmentObjects - Set to true to includes MES response segment objects

- Return

Nothing

- Scope

Client

setParentMESObjectFilter

- Description

Sets the parent MES object to filter the results by.

- Parameters

MESObjectLink parentMESObjectFilter - The parent MES object to filter the results by.

- Return

Nothing

- Scope

Client

setParentMESObjectPath

- Description

Set the path of the parent MES object to filter the results by.

- Parameters

String parentMESObjectPath - The path of the parent MES object to filter the results by.

- Return

Nothing

- Scope

Client

setSelectedMESObject

- Description

Sets the selected MES object link.

- Parameters

MESObjectLink selectedMESObject - The selected MES object link to be set.

- Return

Nothing

- Scope

Client

setSelectedMESObjectTypes

- Description

Sets the MES object types to be selected.

- Parameters

String mesObjectTypes - The MES object types to be selected.

- Return

Nothing

- Scope

Client

setShowEquipmentPath

- Description

Set to True to display equipment paths.

- Parameters

boolean showEquipmentPath - If set to True, the equipment paths are displayed.

- Return

Nothing

- Scope

Client

Extension Functions

objectSelected

- Description

Called when a MES object is selected.

- Parameters

self - A reference to the component that is invoking this function

mesObjectLink - The MESObjectLink that contains a reference to the selected MES object. Use mesObjectLink.getMESObject() to get the MES object itself.

- Return

1

- Scope

Client

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

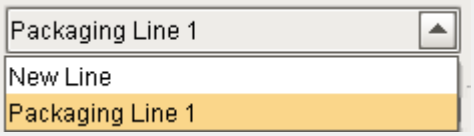
Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

This component does not have any custom properties.

Examples

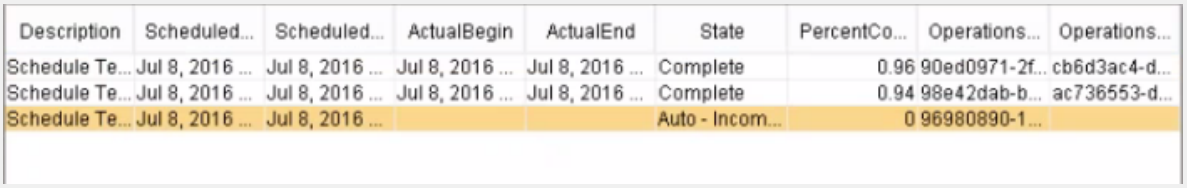
You can select the MES objects from the list. The selected object may be displayed as shown below.



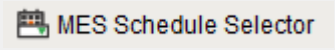
Property Name	Value
Include MES Line Objects	True

MES Schedule Selector

General



Component Palette Icon:



In this Page

Description

MES Schedule Selector is a table component that can be used to view and manage production schedules. The schedule entries may be filtered by Active, Complete or Incomplete lots.

Properties

Name	Scripting	Category	Property Type	Description
Run Control Menu Enabled	runControlMenuEnabled	Behavior	Boolean	Enables the Run Control popup menu by mouse right-click on a row in the table.
Sorting Enabled	sortingEnabled	Behavior	Boolean	Enables automatic multi-column sorting by clicking and CTRL-clicking on the table header.
Can Begin	canBegin	Behavior	Boolean	Read-only. Use in conjunction with a button's enabled property that calls event.source.parent.getComponent('MES Schedule Selector').beginSelected()
Can End	canEnd	Behavior	Boolean	Read-only. Use in conjunction with a button's enabled property that

Name	Scripting	Category	Property Type	Description
				calls event.source.parent.getComponent('MES Schedule Selector').endSelected()
Can Abort	canAbort	Behavior	Boolean	Read-only. Use in conjunction with a button's enabled property that calls event.source.parent.getComponent('MES Schedule Selector').abortSelected()
Column Resize Menu	headerResizeMenus	Behavior	Boolean	Enables a right-click popup menu on the column headers with resizing options.
Column Chooser Menu	headerColumnChooserMenus	Behavior	Boolean	Enables a right-click popup menu on the column

Name	Scripting	Category	Property Type	Description
				headers with options to show and hide columns.
Columns Re-Orderable	columnReorderingAllowed	Behavior	Boolean	Enables the re-ordering of columns by dragging the column headers.
Columns Resizable	columnResizingAllowed	Behavior	Boolean	Enables the resizing of columns by dragging the margins of the column headers.
Auto-Resize Mode	autoResizeMode	Behavior	int	Determines how the table resizes the columns
Row Selection Allowed	rowSelectionAllowed	Behavior	Boolean	This flag is used in conjunction with the Column Selection Allowed flag to determine whether not

Name	Scripting	Category	Property Type	Description
				whole-rows, whole-columns, or both.
Equipment Path	equipmentPath	Data	String	The path to the equipment to show the schedule.
Start Date	startDate	Data	Date	The beginning of the time range to display.
End Date	endDate	Data	Date	The end of the time range to display.
Enable Simultaneous Active	enableSimultaneousActive	Data	Boolean	If true, allows multiple operations to be active at the same time.
Include Manual Incomplete	includeManualIncomplete	Data	Boolean	If true, include manual start incomplete schedule entries in the list.
Include Manual Active	includeManualActive	Data	Boolean	

Name	Scripting	Category	Property Type	Description
				If true, include manual start active schedule entries in the list.
Include Manual Complete	includeManualComplete	Data	Boolean	If true, include manual start completed schedule entries in the list.
Include Auto Incomplete	includeAutoIncomplete	Data	Boolean	If true, include auto start incomplete schedule entries in the list.
Include Auto Active	includeAutoActive	Data	Boolean	If true, include auto start active schedule entries in the list.
Include Auto Complete	includeAutoComplete	Data	Boolean	If true, include auto start completed schedule entries in the list.
Selected Row	selectedRow	Data	int	

Name	Scripting	Category	Property Type	Description
				The index of the first selected row, or -1 if none.
Row Height	rowHeight	Appearance	int	If row resizing is disabled, this will set the height of all rows.
Selection Background	selectionBackground	Appearance	Color	The default background color of selected cells.
Selection Foreground	selectionForeground	Appearance	Color	The default foreground color of selected cells.
Inter Cell Spacing	interCellSpacing	Appearance	Dimension	The space.
Show Horizontal Grid Lines?	showHorizontalLines	Appearance	Boolean	Displays horizontal gridlines making it easier to read.
Show Vertical Grid Lines?	showVerticalLines	Appearance	Boolean	Displays vertical gridlines making it easier to read.
	gridColor	Appearance	Color	

Name	Scripting	Category	Property Type	Description
Grid Line Color				The color used to draw grid lines.
Header Visible	headerVisible	Appearance	Boolean	Allows for hiding of the table's header.
Column Sizing	defaultColumnView	Appearance	String	Represents column sizing and position to preserve user-selected ordering.
Column Attributes Data	columnAttributesData	Appearance	Dataset	The dataset describing the column attributes.
Previous Product Indexed State	previousProductIndexed	Hidden	Boolean	Whether or not the previous product has been indexed to the next cell.

Scripting

Scripting Functions

beginSelected()

- Description

This script function will begin the operation for the currently selected schedule entry. The schedule entries that appear in the MES Schedule Selector component reflect the operations requests that have been created for the equipment specified by the Equipment Path property. Typically, operations requests are created using the MES Schedule View component but can also be created using script functions for importing.

- Parameters

None

- Return

Nothing

- Scope

Client

abortSelected()

- Description

This script function will abort the selected operation for equipment specified by the Equipment Path property. When the operation is aborted, all active segments running underneath it will also be aborted.

- Parameters

None

- Return

Nothing

- Scope

Client

beginNext()

- Description

Based on the scheduled start time, this script function will begin the operation for the next selected schedule entry. The schedule entries that appear in the MES Schedule Selector component reflect the operations requests that have been created for the equipment specified by the Equipment Path property. Typically, operations requests are created using the MES Schedule View component but can also be created using script functions for importing.

- Parameters

None

- Return

Nothing

- Scope

Client

endSelected()

- Description

This script function will end the selected operation for equipment specified by the Equipment Path property. All segments running underneath the selected operation, must be ended prior to calling this script function.

- Parameters

None

- Return

Nothing

- Scope

Client

Extension Functions

configureCell

- Description

Provides a chance to configure the contents of each cell.

- Parameters

self - A reference to the component that is invoking this function.

value - The value in the dataset at this cell.

textValue - The text the table expects to display at this cell (may be overriden by including 'text' attribute in returned dictionary)

selected - A boolean indicating whether this cell is currently selected.

rowIndex - The index of the row in the underlying dataset

colIndex - The index of the column in the underlying dataset

colName - The name of the column in the underlying dataset

rowIndex - The index of the row, as it appears in the table view (affected by sorting)

colView - The index of the column, as it appears in the table view (affected by column re-arranging and hiding)

- Return

Returns a dictionary of name-value pairs with the desired attributes. Availible attributes include: 'background', 'border', 'font', 'foreground', 'horizontalAlignment', 'iconPath', 'text', 'toolTipText', 'verticalAlignment'

You may also specify the attribute 'renderer', which is expected to be a java.swing.JComponent which will be used to render the cell.

- Scope

Client

configureHeaderStyle

- Description

Provides a chance to configure the style of each column header. Return a dictionary of name-value pairs with the designed attributes. Availible attributes include: 'background', 'border', 'font', 'foreground', 'horizontalAlignment', 'toolTipText', 'verticalAlignment'

- Parameters

self - A reference to the component that is invoking this function

colIndex - The index of the column in the underlying dataset

colName - The name of the column in the underlying dataset

- Return

Dictionary of name value pairs

- Scope

Client

initialize

- Description

Called when the window containing this table is opened, or the template containing it is loaded. Provides a change to initialize the table further, for example, selecting a specific row.

- Parameters

self - A reference to the component that is invoking this function

- Return

Nothing

- Scope

Client

onDoubleClick

- Description

Called when the user double-clicks on a table cell.

- Parameters

self - A reference to the component that is invoking this function

rowIndex - Index of the row, starting at 0, relative to the underlying dataset

colIndex - Index of the column starting at 0, relative to the underlying dataset

value - The value at the location clicked on

event - The MouseEvent object that caused this double-click event

- Return

Nothing

- Scope

Client

onPopupTrigger

- Description

Called when the user right-clicks on a table cell. This would be the appropriate time to create and display a popup menu.

- Parameters

self - A reference to the component that is invoking this function

rowIndex - Index of the row, starting at 0, relative to the underlying dataset

colIndex - Index of the column starting at 0, relative to the underlying dataset

value - The value at the location clicked on

event - The MouseEvent object that caused this double-click event

- Return

Nothing

- Scope

Client

beginOperation

- Description

Called before an MES Operaton begins. Return false to prevent the MES Operation from being started.

- Parameters

self - A reference to the component that is invoking this function

MESObjectList - MESObjectList containing MESOperationsPerformance and MESOperationResponse objects. Core and custom properties can be set on the object before the operation begins.

- Return

True

- Scope

Client

endOperation

- Description

Called before an MES Operaton ends. Return false to prevent the MES Operation from being ended.

- Parameters

self - A reference to the component that is invoking this function

MESObjectList - MESObjectList containing MESOperationResponse and any MESResponseSegment objects. The MESResponseSegments objects can be ended in this extension function, which is required before the operation can end.

- Return

True

- Scope

Client

abortOperation

- Description

Called before an MES Operaton is aborted. Return false to prevent the MES Operation from being aborted.

- Parameters

self - A reference to the component that is invoking this function.

MESObjectList - MESObjectList containing MESOperationResponse and any MESResponseSegment objects.

- Return

True

- Scope

Client

requestSelected

- Description

Called after an MES Operaton Request is selected. In this function, the operation request can be started. This allows for operation to automatically start when the user selects a MES Operation Request. Returning false will prevent the new selection.

- Parameters

self - A reference to the component that is invoking this function

mesObjectLink - MESObjectLink object containg the MES Operation Request details. Call mesObjectLink.getMESObject() to get the instance of the MESOperationRequest object.

- Return

True

- Scope

Client

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	

Property	Description
	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

Property	Description
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

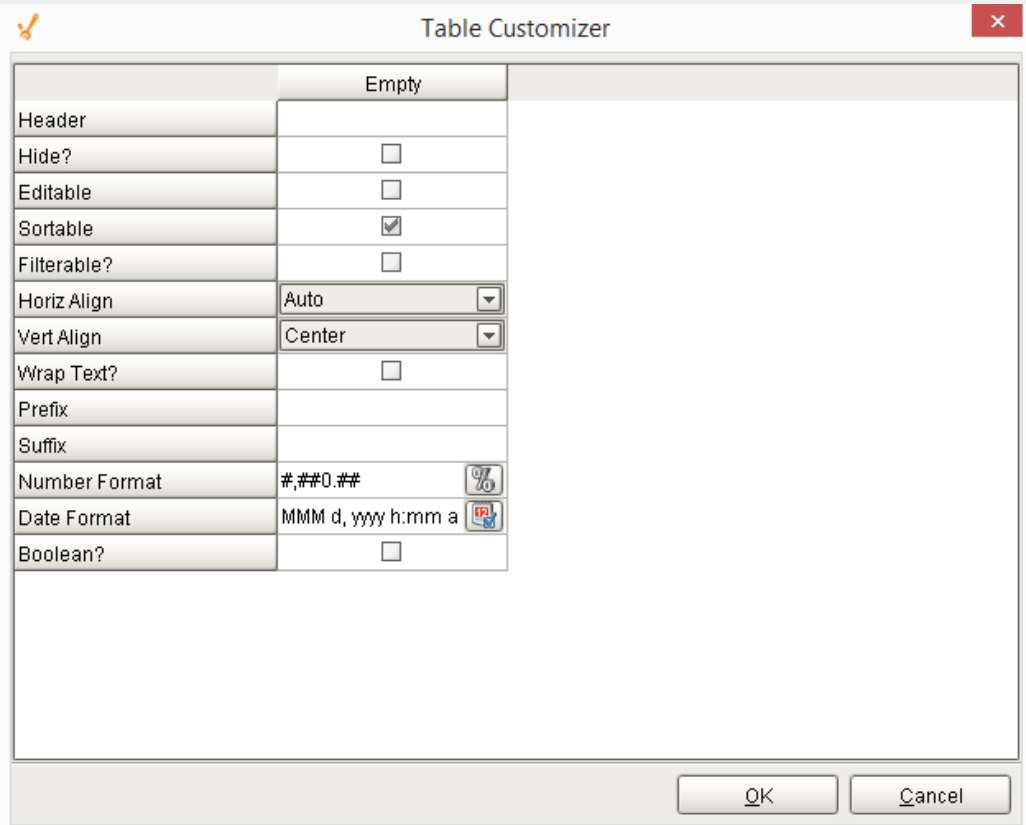
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

Table Customizer

Table Customizer manages the data entered into the MES Schedule Selector. It will allow you to modify the data which is stored inside the MES Schedule Selector. Thus the formatting and alignments are made easy.



Column Configuration

Header - Provide a custom name to the column header.

Hide? - Hides the column

Editable - Allows the editing of the cell pertaining to the column.

Sortable - To make a column filter the data on user's demand .

Sortable - Allows the user to sort the table according to the selected column

Horiz Align - Aligns the contents of the column.

Vert Align - Aligns the contents of the column.

Wrap Text? - Data in the cell wraps to fit the column width. When you change the column width, data wrapping adjusts automatically.

Prefix - A custom text that proceeds the contents of each cell.

Suffix - A custom text that follows the contents of each cell.

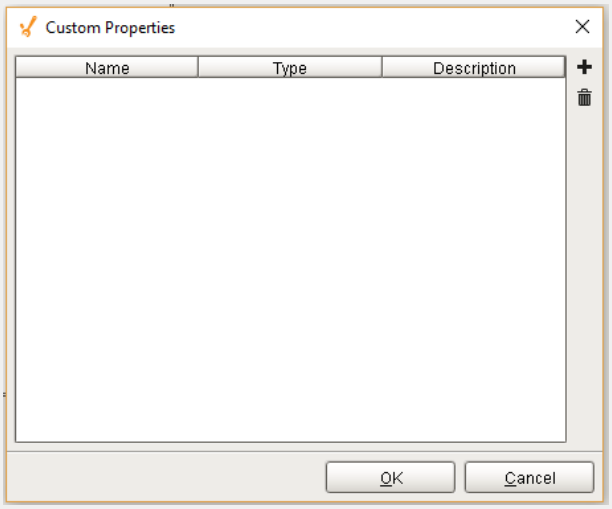
Number Format - A format of the cell is the contents of the cell are number types.

Date Format - Used if the contents of the cell are date types

Boolean? - Changes the contents of the cell to reflect a 'check box' look and feel.

Custom Properties

The custom properties can be used to add user defined properties.



Examples

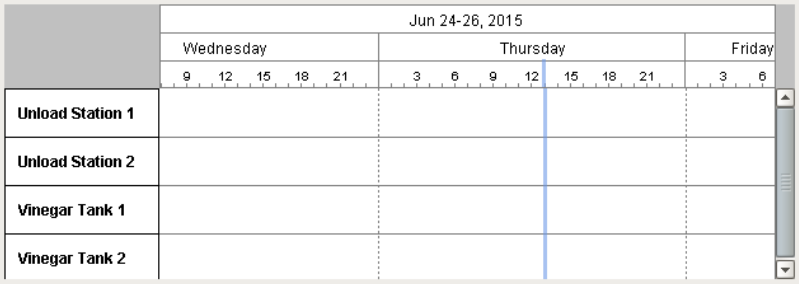
Description	ScheduledBegin	State	OperationsRequestUUID
Receive Material	Aug 5, 2015 8:37 PM	Manual - Incomplete	83c689ab-d374-49b2-8ccd-e7c1...
Receive MaterialA	Aug 5, 2015 8:38 PM	Manual - Incomplete	f9dbb0fb-56a6-4b64-b1cb-2ebd...
Receive Steel (1)	Aug 5, 2015 10:30 PM	Manual - Incomplete	0beebd79-9437-4520-9b27-98c...

Property Name	Value
Start Date	Aug 5, 2015
End Date	Aug 27, 2015
Equipment Path	My Enterprise\California\Receiving\Unload Station 1
Include Manual Incomplete	True

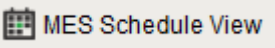
Property Name	Value
Selected Row	2
Selected Request UUID	f9dbb0fb-56a6-4b64-b1cb-2ebd38bfe9a2

MES Schedule View

General



Component Palette Icon:



Info

Drag and Drop Feature

You can drag rows from a power table to the MES Schedule View component. In order to perform drag and drop, you must enable the **Row Dragging Enabled property** of the power table. See the Example section at the bottom of this page for further instructions.

Description

A component that is added to Ignition windows to schedule the MES operations. This is a schedule chart used to schedule the production on equipment. The date and time may be set for each equipment.

Properties

Name	Scripting	Category	Property Type	Description
Time Resolution	timeResolution	Behavior	Int4	Controls the resolution.
Past Scheduling Enabled	pastSchedulingEnabled	Behavior	Boolean	If true, allow scheduling in the past.
Overlapping Enabled	overlappingEnabled	Behavior	Boolean	If true, allow scheduling enteries to overlap.
Start Date	startDate	Data	DateTime	The beginning of the time range to display.
End Date	endDate	Data	DateTime	The end of the time range to display.
Show Categories	showCategories	Data	String	Schedule categories to display in the schedule. If blank, then Active category is used by default. There can be user

Name	Scripting	Category	Property Type	Description
				defined categories, but Active is reserved.
Edit Category	editCategory	Data	String	Schedule category that can be edited within the schedule component. If blank, then Active category is used by default. There can be user defined categories, but Active is reserved.
Show Equipment Path	showEquipmentPath	Data	Boolean	If true, it will show equipment names with equipment paths.
Excluded Equipment Path	excludedEquipmentPath	Data	Boolean	Comma separated string list to be excluded from equipment paths.
Name Filter	nameFilter	Data	String	

Name	Scripting	Category	Property Type	Description
				Filter value, including * and ? wildcard characters, to filter results by MES object names.
Include MES Lines	includeMESLines	Data	Boolean	If true, including MES equipment class objects.
Include MES Storage Units	includeMESStorageUnits	Data	Boolean	If true, including MES equipment class objects.
Schedule Background	scheduleBackground	Appearance	Color	The background color of the schedule area.
Restriction Background	restrictionBackground	Appearance	Color	The background color of restricted.
Line Color	lineColor	Appearance	Color	The color of lines in the schedule view.
Now Line Color	nowLineColor	Appearance	Color	The color of line indicating the current date and time.
Item Height	itemHeight	Appearance	int	The equipment item row height.
Item Font	itemFont	Appearance	Font	

Name	Scripting	Category	Property Type	Description
				The font to use for equipment item names.
Active Item Background	activeItemBackground	Appearance	Color	The background color of equipment items that are active.
Active Item Foreground	activeItemForeground	Appearance	Color	The foreground color of equipment items that are active.
Active Item Status Icon Path	activeItemStatusIconPath	Appearance	String	The path to an Ignition image to use for equipment items that are active.
Inactive Item Background	inactiveItemBackground	Appearance	Color	The background color of equipment items that are inactive.
Inactive Item Foreground	inactiveItemForeground	Appearance	Color	The foreground color of equipment items that are inactive.
	inactiveItemStatusIconPath	Appearance	String	

Name	Scripting	Category	Property Type	Description
Inactive Item Status Icon Path				The path to an Ignition image to use for equipment items that are inactive.
Show Progress Bar	showProgressBar	Appearance	Boolean	If set to true show the progress bar for schedule entries. The percent complete value must also greater than 0 for the bar to be displayed for a schedule entry.
Progress Background	progressBackground	Appearance	Color	The background color of schedule entry progress boxes.
Progress Border Color	progressBorderColor	Appearance	Color	The border color of schedule entry progress boxes.
Progress Fill Color	progressFillColor	Appearance	Color	The fill color of schedule entry progress boxes.
Entry Configuration	entryConfiguration	Appearance	DataSet	

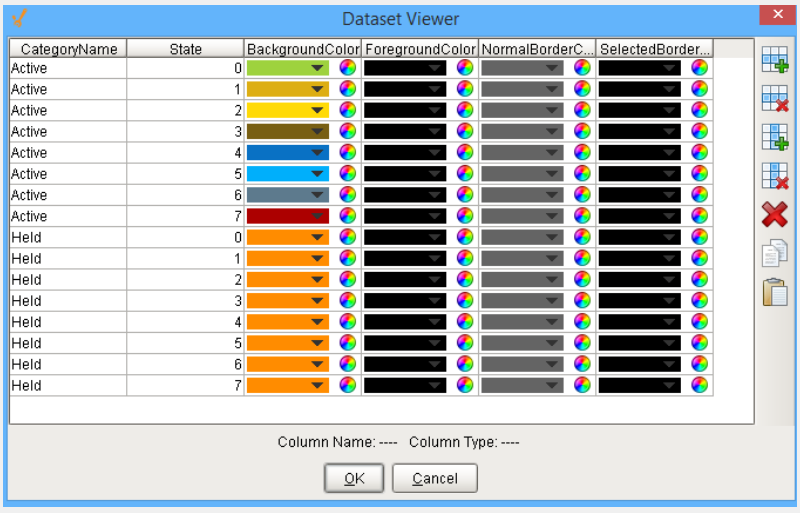
Name	Scripting	Category	Property Type	Description
				A dataset that stores entry appearance configuration.
User Menu Items	userMenuItems	Behavior	DataSet	A dataset that stores user menu items.

Entry Configuration Property

This property controls the appearance of each entry. Click on the **Dataset Viewer** icon for the **Appearance** property in **Property Editor** to set the values.

For each value you enter, a state is created. For example, you can see below that there are 16 values in the Dataset Viewer which corresponds to the 16 category and state combinations on the Schedule View component. Each state belongs to a category, the categories inbuilt on Ignition are Active and Actual. The Active includes the schedule that is currently active and the Actual includes those which have completed. You can add your own categories, 'Held' is added in the given example.

Color of the entry is determined by BackgroundColor. ForegroundColor will decide the color of the text. NormalBorderColor is the boundary color for an unselected state. Color for the border of selected state is given by the SelectedBorderColor.



Below is a table that describes the states of the entry configuration.

State	Description
0	Selected in editing mode
1	Auto Incomplete
2	Auto Running
3	Auto Complete
4	Manual Incomplete
5	Manual Running
6	Manual Complete
7	Faulted

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

configureScheduleBlock

- Description

Called before displaying a schedule entry block. This can be used to change the appearance of a schedule entry block.

- Parameters

self - A reference to the component that is invoking this function.

mesScheduleEntry - The MESScheduleEntry object associated with the schedule block being rendered. Nothing should be changed in the object. See MESScheduleEntry in the MES documentation for more information.

settings - The default display setting that can be changed to alter the appearance of the schedule block. To change the background use settings.setBackground('Orange') or settings.setBackground('#345a2f').

- Return

Nothing

- Scope

Client

showEditor

- Description

Called before showing the editor panel. This can be used to add the functionality of, the built-in schedule entry editor panel. To prevent the built-in editor panel from appearing, return False.

- Parameters

self - A reference to the component that is invoking this function

scheduleItem - The ScheduleItem object contains information about the equipment being scheduled. It also includes schedule restrictions and current schedule entries for the equipment and time period specified by the component.

- Return

True

- Scope

Client

hideEditor

- Description

Called after hiding the editor panel. This can be used to add functionality following that of the built-in schedule entry editor panel.

- Parameters

self - A reference to the component that is invoking this function.

scheduleItem - The MESObjectLink that contains a reference to the selected MES object. Use mesObjectLink.getMESObject() to get the MES object itself.

- Return

True

- Scope

Client

updateMenuItem

- Description

Called for each menu item. The enabled state or other menu item setting can be changed.

- Parameters

self - A reference to the component that is invoking this function.

menuItem - The menu item object that can be modified.

mesScheduleEntry - The MESScheduleEntry object associated with the currently selected schedule entry.

- Return

True

- Scope

Client

onSave

- Description

Called before saving a new or modified schedule.

- Parameters

self - A reference to the component that is invoking this function.

mesObjectList - The MESObjectList that contains references to all MES objects for the operations schedule.

- Return

True

- Scope

Client

onSaveError

- Description

Called if an error occurs while saving a new or modified schedule. Return true to cancel the changes and close the edit panel.

- Parameters

self - A reference to the component that is invoking this function.

mesObjectList - The MESObjectList that contains references to all MES objects for the operations schedule.

errorMessage - The error mesage describing the error.

- Return

False

- Scope

Client

getToolTipText

- Description

Called to get tool tip text for a restricted time or schedule entry on the schedule view.

- Parameters

self - A reference to the component that is invoking this function.

entryType - The type name as a string of the entryObject.

entryObject - Either a MESScheduleRestriction or a MESScheduleEntry object that the tool tip text is needed.

- Return

Nothing

- Scope

Client

onRowsDropped

Read **Example 2** below in the Examples section.

- Description

Called when the user has dropped rows from the Ignition power table component. The source table must have dragging enabled.

- Parameters

self - A reference to the component that is invoking this function

sourceTable - A reference to the table that the rows were dragged from.

rows - An array of the row indices that were dragged, in the order they were selected.

rowData - A dataset containing the rows that were dragged.

equipmentPath - The path for the equipment item that the user dropped on.

category - The schedule category that the user dropped on.

dateTime - The date and time at the point where the drop occured.

- Return

Nothing

- Scope

Client

Event Handlers

menu

userMenuItemClicked

When the user right clicks on the schedule component, a popup menu is displayed. Custom user menu items can be added to this popup menu by adding rows to the **User Menu Items** dataset.

The UserMenuItemClicked event is fired when the menu item is clicked, or if the user selects the menu item using the keyboard and presses the Enter key. It can also occur if an access key or shortcut key is pressed that is associated with the MenuItem.

Tip

See the tech note on Adding Custom User menu Items to MES Schedule View

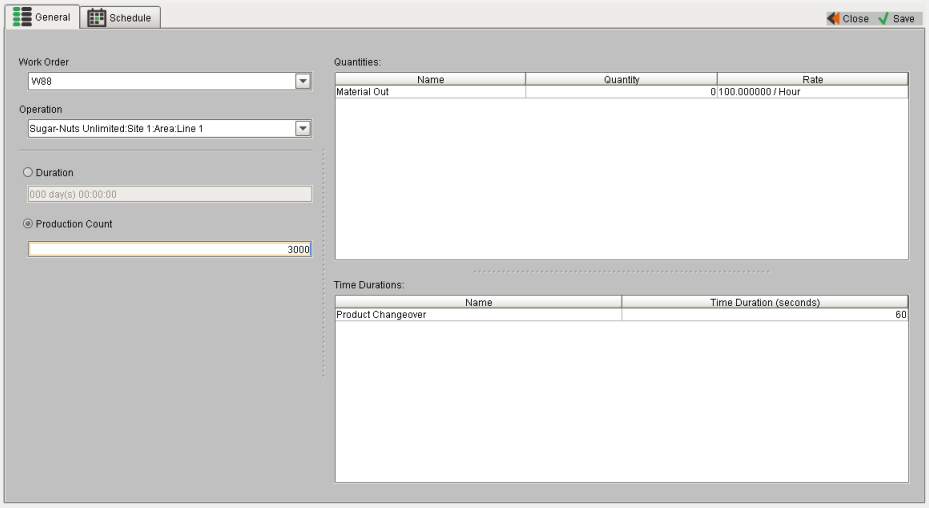
Property	Description
source	The component that fired this event.
getMenuItemName()	Returns the name of the selected menu item.
getScheduleEntry()	Returns the schedule entry object containing details about the schedule category, begin, actual begin and end, state, etc.

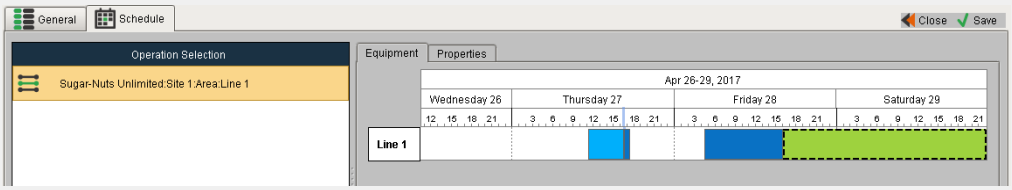
Code Example 1

Code Snippet

```
if(event.getMenuItemName() == 'Hold'):
    uuid = event.getScheduleEntry().
getMESOperationsScheduleLink().getMESObjectUUID()
    system.mes.changeScheduleCategory(uuid, 'Held')
elif(event.getMenuItemName() == 'Release Hold'):
    uuid = event.getScheduleEntry().
getMESOperationsScheduleLink().getMESObjectUUID()
    system.mes.changeScheduleCategory(uuid, 'Active')
print event.getScheduleEntry()
```

Output






```
Category: Held, schedule begin: 2016-08-10 12:39:29.0
, schedule end: 2016-08-11 12:40:29.0, actual begin:
null, actual end: null, state: 0
```

Code Example 2

Code Snippet

```
if event.getMenuItemName() == "Details":
 se = event.getScheduleEntry()
 print se.getScheduledStartDate()
 print se.getScheduledEndDate()
 print se.getActualStartDate()
 print se.getActualEndDate()
 print se.getPctDone()
 print se.getCategory()
 print se.getState()
 print se.getStateAsString()
 if se.hasMESOperationsScheduleLink():
 print se.getMESOperationsScheduleLink()
 print se.getMESOperationsRequestLink()
 if se.hasMESOperationsResponseLink():
 print se.getMESOperationsResponseLink()
```

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

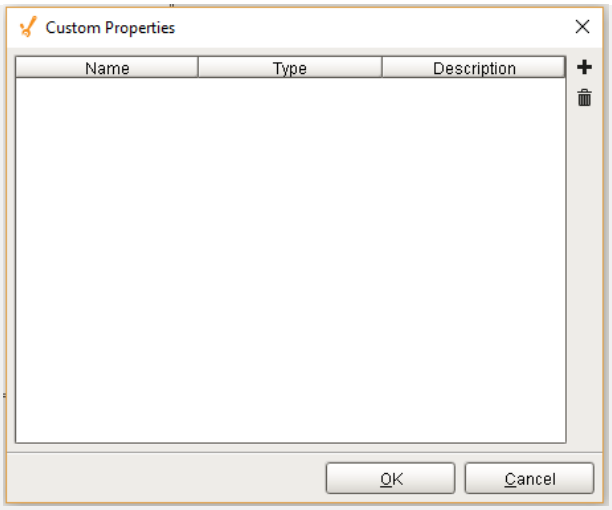
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: Remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

Custom Properties

The custom properties can be used to add user defined properties.

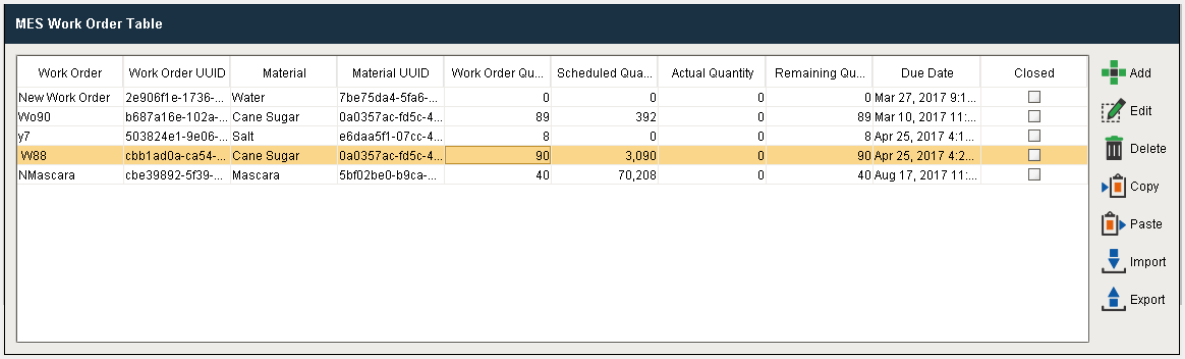


Example 1

Dragging the work order to the schedule view component

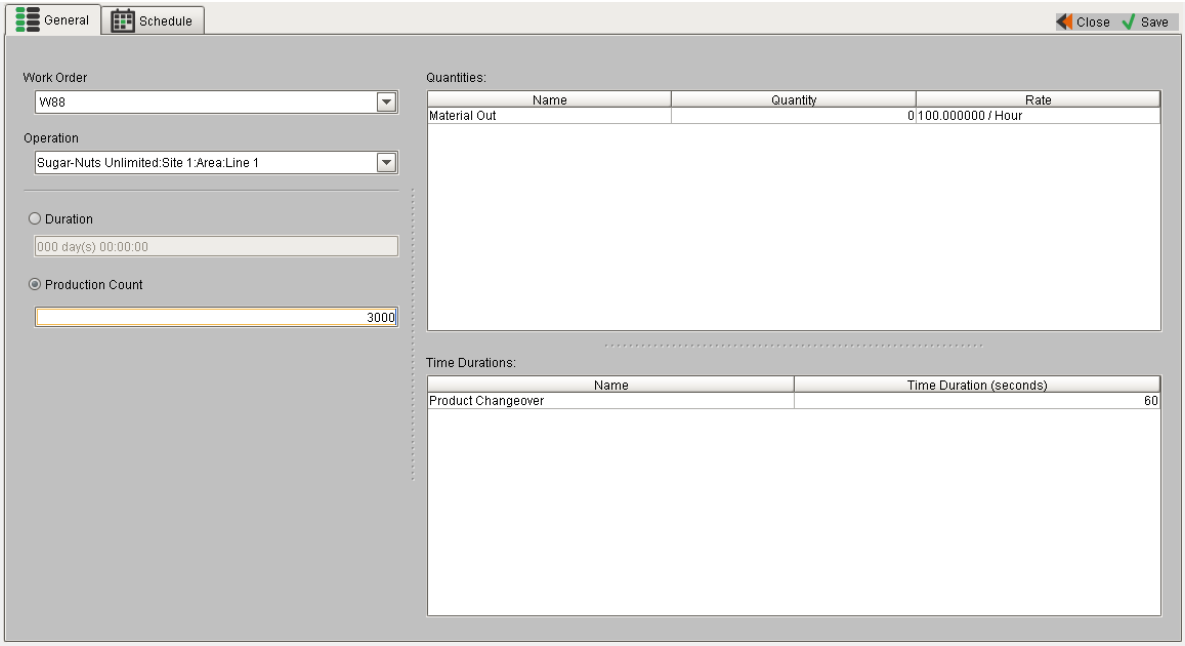
Step 1

Set the Row Dragging Enabled property of the Work Order Table to True and select the desired row.



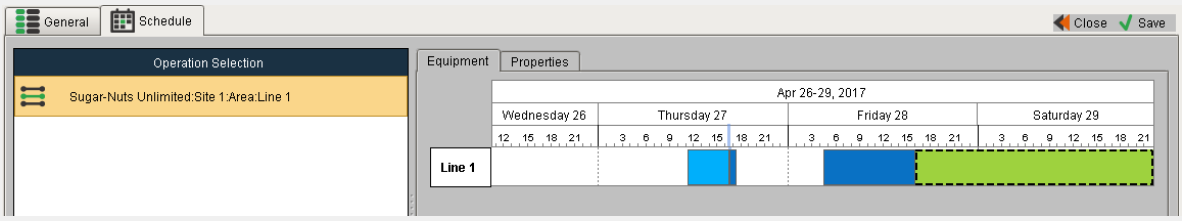
Step 2

Drag the selected row to the schedule view component. Make sure you release it at the appropriate equipment item. In this example Line 1 production item is configured for Cane Sugar. A scheduling window will appear as shown below.

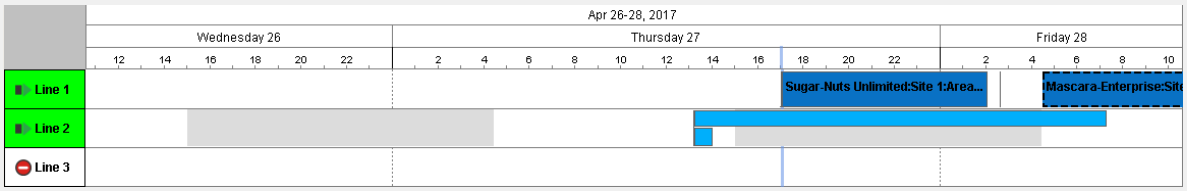


Step 3

Provide the duration or the production count of the schedule. Select the schedule tab to view the new schedule. Click Save.

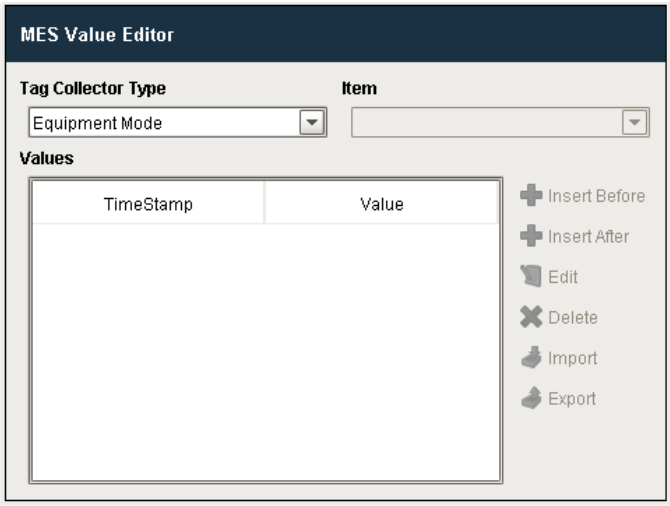


The schedule you just created will be displayed in the MES Schedule View component as shown.



MES Value Editor

General



Component Palette Icon:

MES Value Editor

In this Page

See Adjusting Production Run Data page for various MES Value Editor settings.

Description

MES Value Editor component is used to change values that were automatically captured by the system but need to be modified, for example machine said it was in production mode, but it was actually in maintenance mode, or production counts are off.

Properties

Name	Scripting	Category	Property Type	Description

Equipment Path	equipmentPath	Data	String	The path to the equipmen to show the schedule. ⚠ If you copy and paste an equipment path to the MES Value Editor component Equipment Path property, the value does not persist after saving, closing the window and re-opening it. This is by design. Otherwise everytime a window is opened, it will cause calls to the server which are not needed. ✓ Bindings do persist, the property be bound to a tag or a binding to a root container custom property containing the equipment

Name	Scripting	Category	Property Type	Description
				path gets around this issue.
Start Date	startDate	Data	Date	The beginning of the time range to display.
End Date	endDate	Data	Date	The end of the time range display.
Column Attributes Data	columnAttributesData	Appearance	Dataset	The dataset describing the column attributes.
Data	data	Data	Dataset	The data for this table.
Selected Row	selectedRow	Data	int	The index of the first selected row, or -1 if none
Title Font	titleFont	Appearance	Font	The font of text of the title bar.
Title Foreground Color	titleForeground	Appearance	Color	The foreground color of th title bar.
Title Background Color	titleBackground	Appearance	Color	The background color of t title bar.
Header Font	headerFont	Appearance	Font	The font of text of the tabl header.
	headerForeground	Appearance	Color	The foreground color of th table header.

Name	Scripting	Category	Property Type	Description
Header Foreground Color				
Row Font	rowFont	Appearance	Font	The font of text of the tabl header.
Row Foreground Color	rowForeground	Appearance	Color	The foreground color of rows in the table.
Row Background Color	rowBackground	Appearance	Color	The background color of rows in the table.
Row Selection Foreground Color	selectionForeground	Appearance	Color	The default foreground co of selected cells.
Row Selection Background Color	selectionBackground	Appearance	Color	The default background color of selected cells.
Auto Row Height Enabled	autoRowHeightEnabled	Appearance	Boolean	If true, the row height of th table will be adjusted automatically.
Row Height	rowHeight	Appearance	int	If row resizing is disabled, this will set the height of a rows.
Show Horizontal Grid Lines?	showHorizontalLines	Appearance	Boolean	Displays horizontal gridlin making it easier to read.

Name	Scripting	Category	Property Type	Description
Show Vertical Grid Lines?	showVerticalLines	Appearance	Boolean	Displays vertical gridlines making it easier to read.
Grid Line Color	gridColor	Appearance	Color	The color used to draw gr lines.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

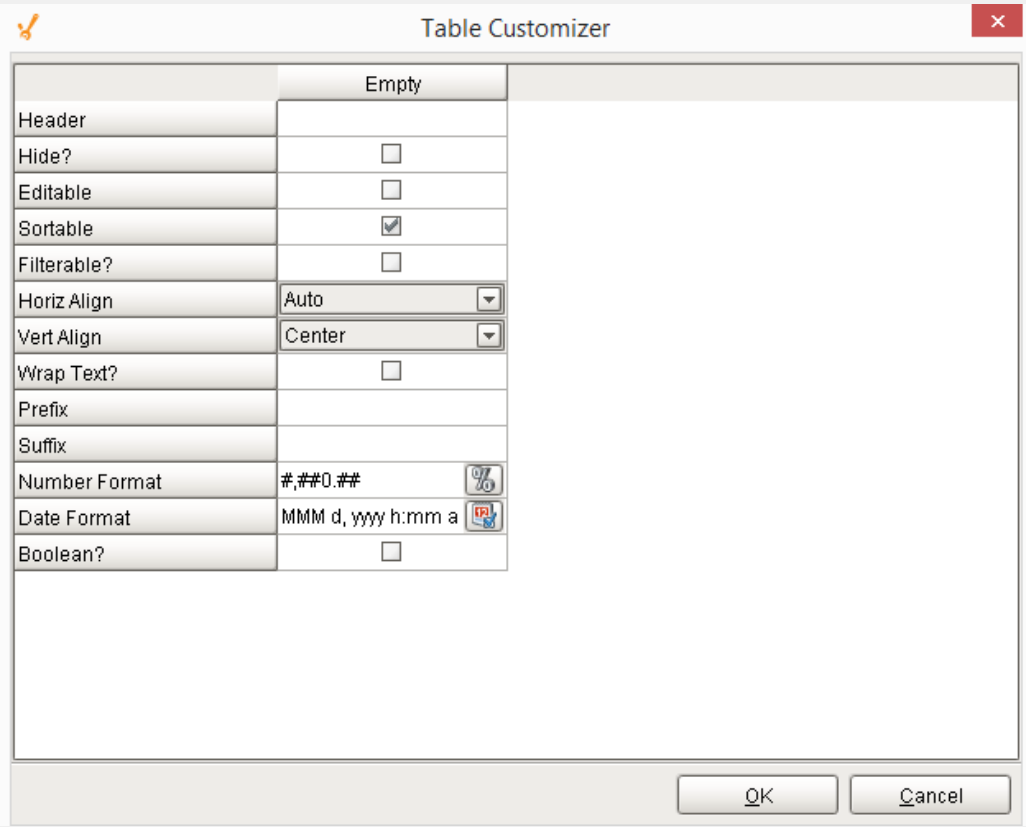
Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.

Property	Description
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

Table Customizer

Table Customizer manages the data entered into the MES Schedule Selector. It will allow you to modify the data which is stored inside the MES Schedule Selector. Thus the formatting and alignments are made easy.



Column Configuration

Header - Provide a custom name to the column header.

Hide? - Hides the column

Editable - Allows the editing of the cell pertaining to the column.

Sortable - To make a column filter the data on user's demand .

Sortable - Allows the user to sort the table according to the selected column

Horiz Align - Aligns the contents of the column.

Vert Align - Aligns the contents of the column.

Wrap Text? - Data in the cell wraps to fit the column width. When you change the column width, data wrapping adjusts automatically.

Prefix - A custom text that proceeds the contents of each cell.

Suffix - A custom text that follows the contents of each cell.

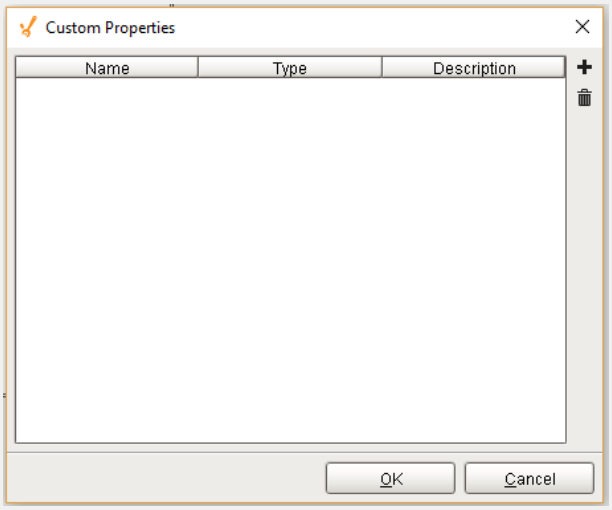
Number Format - A format of the cell is the contents of the cell are number types.

Date Format - Used if the contents of the cell are date types

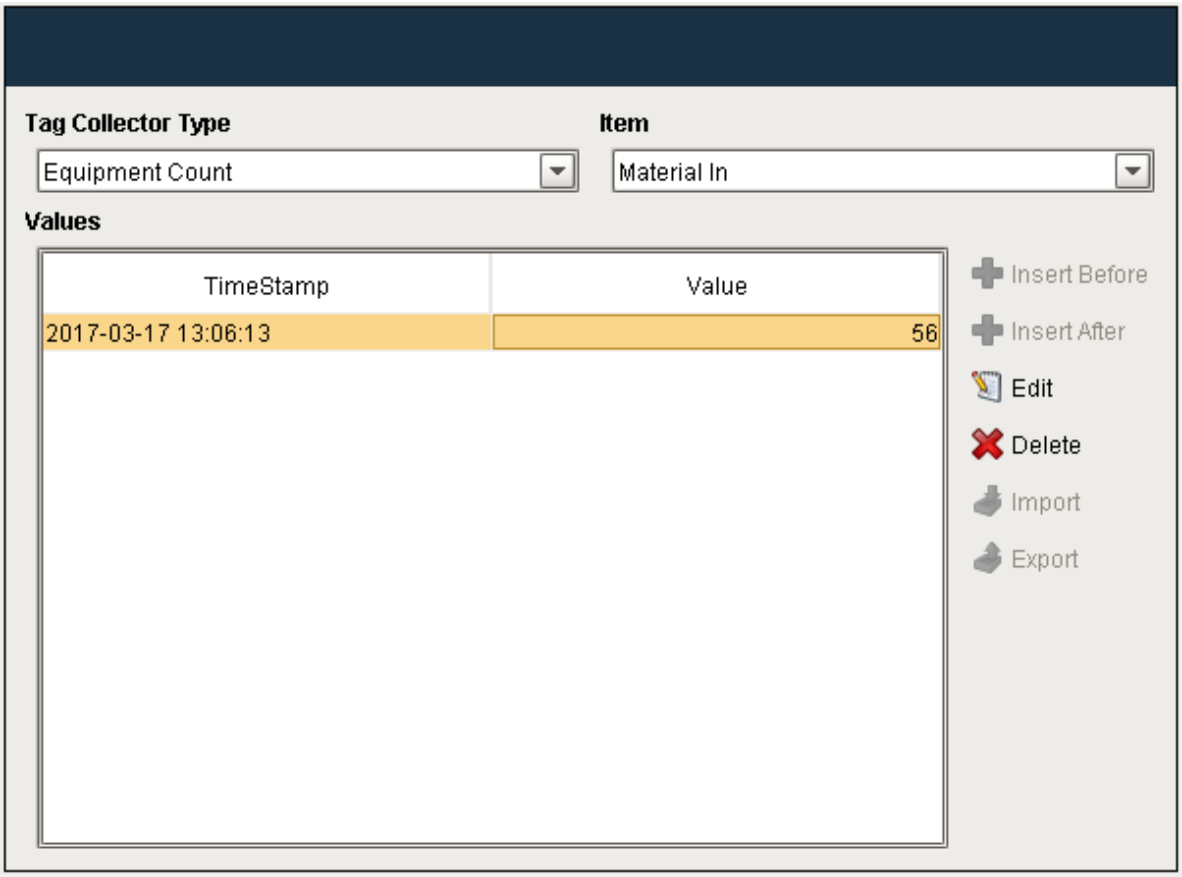
Boolean? - Changes the contents of the cell to reflect a 'check box' look and feel.

Custom Properties

The custom properties can be used to add user defined properties.



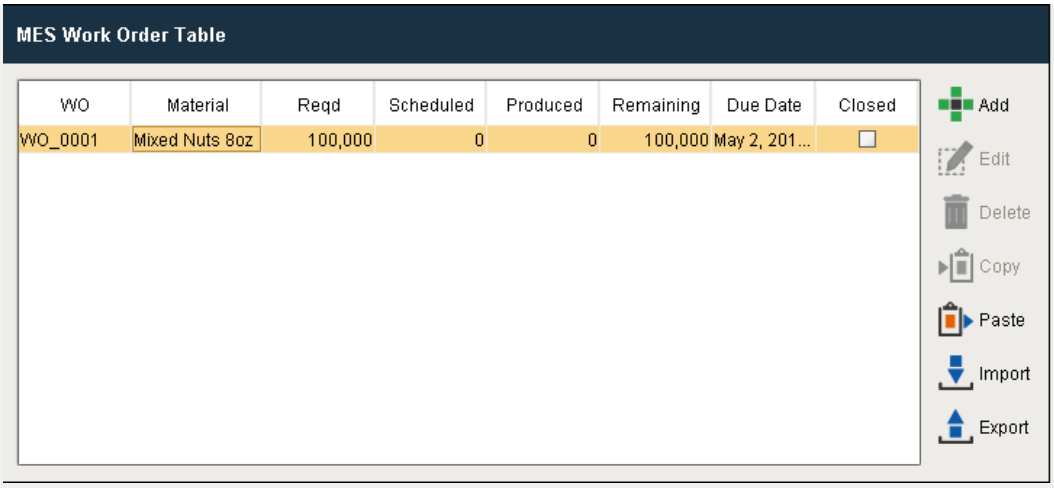
Examples



Property Name	Value
Equipment Path	[global]\Enterprise\San Marcos\MP Rotator\MP Rotator 1

MES Work Order Table

General



Component Palette Icon: MES Work Order Table

See Creating and Managing Work Orders page for various MES Work Order Table settings.

Info

Drag and Drop Feature

You can drag rows from a power table to the MES Work Order Table component. In order to perform drag and drop, you must enable the **Row Dragging Enabled property** of the power table.

Description

A component that displays all the available work orders in a table and calculates the units produced, scheduled and remaining for each work order. All work orders are automatically displayed from the "WorkOrder" database table within the date range of From Date and To Date properties without the need for custom SQL statements or script.

Properties

Name	Scripting	Category	Property Type	Description
Material Name Filter	materialNameFilter	Data	String	The material name filter, that can include * and ? wildcard characters, to filter the work orders by.
Work Order Name Filter	workOrderNameFilter	Data	String	The work order name filter, that can include * and ? wildcard characters, to filter the work orders by.
Equipment Path Filter	equipmentPathFilter	Data	String	The equipment path filter, that can include * and ? wildcard characters, to filter the work orders by.
Closed Start Date	closedStartDate	Data	Date	The start date to get work orders for if Show Closed Work Order property is true.
Closed End Date	closedEndDate	Data	Date	The end date to get work orders for if Show Closed Work Order property is true.
	showClosedWorkOrder	Data	Boolean	

Name	Scripting	Category	Property Type	Description
Show Closed Work Order				If true, show closed work orders.
Column Attribute Data	columnAttributesData	Data	Dataset	The dataset describing the data attributes.
Data	data	Data	Dataset	The data for this table.
Selected Row	selectedRow	Data	int	The index of the first selected row, or -1 if none.
Row Dragging Enabled	rowDragEnabled	Behavior	Boolean	If true, allows a row to dragged from this component to others.
Font	font	Appearance	Font	Font of text of this component.
Foreground Color	foreground Color	Appearance	Color	The foreground color of the component.
Background Color	backgroundColor	Appearance	Color	The background color of the component.
Title Font	titleFont	Appearance	Font	The font of text of the title bar.
Title Foreground Color	titleForeground	Appearance	Color	The foreground color of the title bar.

Name	Scripting	Category	Property Type	Description
Title Background Color	titleBackground	Appearance	Color	The background color of the title bar.
Header Font	headerFont	Appearance	Font	The font of text of the table header.
Header Foreground Color	headerForeground	Appearance	Color	The foreground color of the table header.
Row Font	rowFont	Appearance	Font	The font of text of the table header.
Row Foreground Color	rowForeground	Appearance	Color	The foreground color of rows in the table.
Row Background Color	rowBackground	Appearance	Color	The background color of rows in the table.
Row Selection Foreground Color	selectionForeground	Appearance	Color	The foreground color of a selected row in the table.
Row Selection Background Color	selectionBackground	Appearance	Color	The background color of a selected row in the table.
Auto Row Height Enabled	autoRowHeightEnabled	Appearance	Boolean	If true, the row height of the table will be adjusted automatically.

Name	Scripting	Category	Property Type	Description
Row Height	rowHeight	Appearance	int	The height of each row of the table.
Grid Line Color	gridColor	Appearance	Color	The color of grid lines in the table.
Show Horizontal Grid Lines	showHorizontalLines	Appearance	Boolean	Determines whether horizontal grid lines are shown in the table.
Show Vertical Grid Lines	showVerticalLines	Appearance	Boolean	Determines whether vertical grid lines are shown in the table.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

initialize

- Description

Called when the window containing this table is opened, or the template containing it is loaded. Provides a chance to initialize the table further, for example, selecting a specific row.

- Parameters

self - A reference to the component that is invoking this function.

- Return

Nothing

- Scope

Client

isRowEditable

- Description

Returns a boolean that determines whether or not the current row is editable.

- Parameters

self - A reference to the component that is invoking this function.

mesWorkOrder - The MESWorkOrder object itself.

- Return

True

- Scope

Client

isRowDeletable

- Description

Returns a boolean that determines whether or not the current row is deletable.

- Parameters

self - A reference to the component that is invoking this function.

mesWorkOrder - The MESWorkOrder object itself.

- Return

True

- Scope

Client

onRowEdited

- Description

Called when the user has edited a row in the table.

- Parameters

self - A reference to the component that is invoking this function.

mesWorkOrder - The MESWorkOrder object itself.

- Return

None

- Scope

Client

onRowDeleted

- Description

Called when the user has deleted a row in the table.

- Parameters

self - A reference to the component that is invoking this function.

mesWorkOrder - The MESWorkOrder object itself.

- Return

None

- Scope

Client

onColumnsCreate

- Description

Called when columns are created in the table. Provides a chance to add custom columns to the table.

- Parameters

self - A reference to the component that is invoking this function.

- Return

Returns a dictionary of custom column name-type pairs.

- Scope

Client

onRowAdd

- Description

Called when a row is added in the table. Provides a chance to insert values to custom columns in the table.

- Parameters

self - A reference to the component that is invoking this function.

mesWorkOrder - The MESWorkOrder object itself.

- Return

Returns a dictionary of custom column name-value pairs.

- Scope

Client

onMousePress

- Description

Called when the user clicks on a table cell.

- Parameters

self - A reference to the component that is invoking this function.

rowIndex - Index of the row, starting at 0, relative to the underlying dataset.

colIndex - Index of the column starting at 0, relative to the underlying dataset.

colName - Name of the column in the underlying dataset.

value - The value at the location clicked on.

event - The MouseEvent object that caused this press event.

- Return

Nothing

- Scope

Client

onDoubleClick

- Description

Called when the user double-clicks on a table cell.

- Parameters

self - A reference to the component that is invoking this function.

rowIndex - Index of the row, starting at 0, relative to the underlying dataset.

colIndex - Index of the column starting at 0, relative to the underlying dataset.

colName - Name of the column in the underlying dataset.

value - The value at the location clicked on.

event - The MouseEvent object that caused this double-click event.

- Return

Nothing

- Scope

Client

onPopupTrigger

- Description

Called when the user right-clicks on a table cell. This would be the appropriate time to create and display a popup menu.

- Parameters

self - A reference to the component that is invoking this function.

rowIndex - Index of the row, starting at 0, relative to the underlying dataset.

colIndex - Index of the column starting at 0, relative to the underlying dataset.

colName - Name of the column in the underlying dataset.

value - The value at the location clicked on.

event - The MouseEvent object that caused this popup trigger event.

- Return

Nothing

- Scope

Client

onRowsDropped

- Description

Called when the user has dropped rows on this table. Note that the rows may have come from this table or another table. The source table must have dragging enabled.

- Parameters

self - A reference to the component that is invoking this function

sourceTable - A reference to the table that the rows were dragged from. Will be equal to 'self' if the rows were dragged and dropped in the same table.

rows - An array of the row indices that were dragged, in the order they were selected.

rowData - A dataset containing the rows that were dragged.

dropIndexLocation - Row index where the rows were dropped.

- Return

Nothing

- Scope

Client

Code Example

```
    if sourceTable.getName() == 'ERP Work Order Table':
#Name of the Power table that the Work Orders come from
        for row in range(rowData.rowCount):
            #Assuming this is the order of data in the
ERP Power Table
            woName = rowData.getValueAt(row,0)
            prodCode = rowData.getValueAt(row,1)
            qty = rowData.getValueAt(row,2)
            dueDate = rowData.getValueAt(row,3)
            #Check if Work order already exists before
we create it again
            woFilter = system.mes.workorder.
createMESWorkOrderFilter()
            woFilter.setWorkOrderNameFilter(woName)
            results = system.mes.workorder.
getMESWorkOrders(woFilter)
            if len(results) == 0:
                #Given a work order name, create the
work order and then save the work order to manifest the
change.
                matLink = system.mes.
getMESObjectLinkByName('MaterialDef', prodCode)
                woObj=system.mes.workorder.
createMESWorkOrder(woName, matLink)
                #Set some production related properties
                woObj.setWorkOrderQuantity(qty)
                woObj.setDueDate(dueDate)
                system.mes.saveMESObject(woObj)
            else:
                system.gui.messageBox(woName + " already
exists")
```

Event Handlers

propertyChange

propertyChange

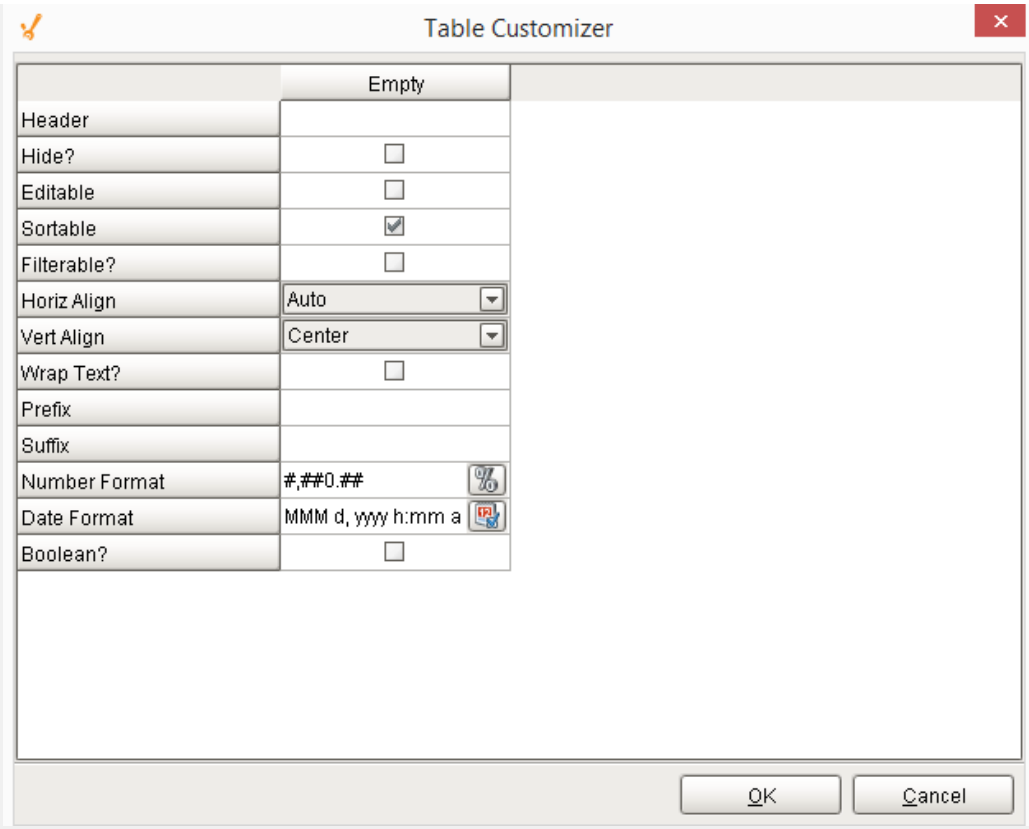
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

Table Customizer

Table Customizer manages the data entered into the MES Schedule Selector. It will allow you to modify the data which is stored inside the MES Schedule Selector. Thus the formatting and alignments are made easy.



Column Configuration

Header - Provide a custom name to the column header.

Hide? - Hides the column

Editable - Allows the editing of the cell pertaining to the column.

Sortable - To make a column filter the data on user's demand .

Sortable - Allows the user to sort the table according to the selected column

Horiz Align - Aligns the contents of the column.

Vert Align - Aligns the contents of the column.

Wrap Text? - Data in the cell wraps to fit the column width. When you change the column width, data wrapping adjusts automatically.

Prefix - A custom text that proceeds the contents of each cell.

Suffix - A custom text that follows the contents of each cell.

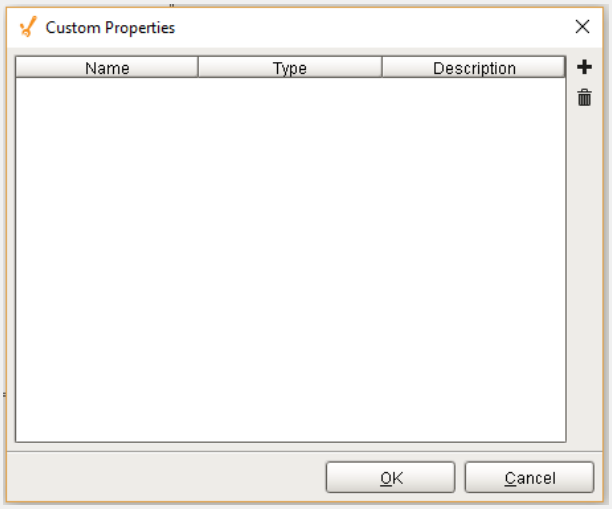
Number Format - A format of the cell is the contents of the cell are number types.

Date Format - Used if the contents of the cell are date types

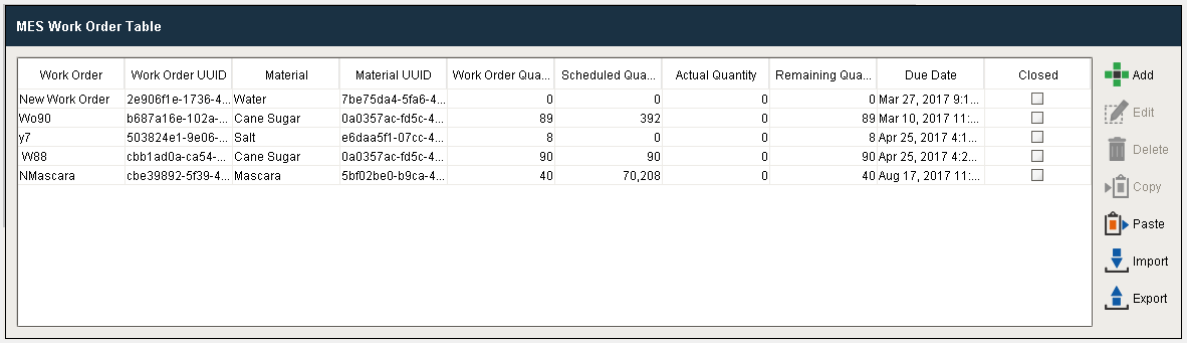
Boolean? - Changes the contents of the cell to reflect a 'check box' look and feel.

Custom Properties

The custom properties can be used to add user defined properties.



Examples



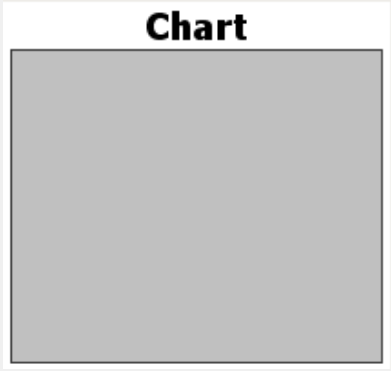
Work Order Table

The users can click on a checkbox in the Closed column to close out a work order. After it is closed out, it will no longer show in the Work Order Table component and it will not be available in any other work order selector components. This feature is provided because some production runs may finish before the target number of units are produced due to lack of raw materials, change in production priorities, etc.

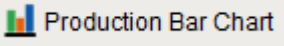
The user can also click on a checkbox in the Hide column to hide the work order from being shown in the Work Order Component. Implementations that integrate with other software systems, such as an ERP system, may show work orders that are not relevant to this system. By hiding them, this list can be kept clean of unrelated work orders.

Production Bar Chart

General



Component Palette Icon:



Description

A component that displays a pie chart with drill down capabilities. This extends from the Bar Chart Component  that comes with Ignition.

Properties

Name	Scripting	Category	Property Type	Description
Data	data	Data	DataSet	The data driving the chart.
Drill Down Options	drillDownOptions	Data	DataSet	Dataset with drill down options.
	previousDrillDownEnabled	Data	Boolean	

Name	Scripting	Category	Property Type	Description
Previous Drill Down Enabled				If true, show previous in drill down menu.
Chart Title	title	Appearance	String	An optional title that will appear at the top of the chart.
Value Axis Label	valueLabel	Axes	String	The label for the value axis.
Value Axis Label format	valueLabelFormat	Axes	String	The label format for the value axis.
Category Axis Label	categoryLabel	Axes	String	The label for the category axis.
Value Axis Auto-Range	valAxisAutoRange	Axes	Boolean	If true, the value axis range will be determined automatically. If false, the specified upper and lower bounds will be used.
Value Axis Lower Bound	valAxisLowerBound	Axes	Float8	The lower bound of the value axis. Used only when auto-range is false.
Value Axis Upper Bound	valAxisUpperBound	Axes	Float8	

Name	Scripting	Category	Property Type	Description
				The upper bound of the value axis. Used only when auto-range is false.
Title Font	titleFont	Axes	Font	The font for the chart's title.
Legend Font	legendFont	Axes	Font	The font for the legend items.
Bar Label Font	barLabelFont	Axes	Font	The font for the bar labels.
Bar Label Offset	barLabelOffset	Axes	Float8	The offset between the bar and the bar label.
Value Axis Label Font	valAxisLabelFont	Axes	Font	The font for the value axis label.
Category Axis Label Font	catAxisLabelFont	Axes	Font	The font for the category axis label.
Value Axis Tick Font	valAxisTickFont	Axes	Font	The font for the value axis' ticks.
Category Axis Tick Font	catAxisTickFont	Axes	Font	The font for the category axis' ticks.
Bar Label Color	barLabelColor	Axes	Color	The color for the bar labels.

Name	Scripting	Category	Property Type	Description
Value Axis Label Color	valAxisLabelColor	Axes	Color	The color for the value axis label.
Category Axis Label Color	catAxisLabelColor	Axes	Color	The color for the category axis label.
Value Axis Tick Color	valAxisTickColor	Axes	Color	The color for the value axis' ticks.
Category Axis Tick Color	catAxisTickColor	Axes	Color	The color for the category axis' ticks.
Value Axis Upper Margin	valAxisUpperMargin	Axes	Float8	The upper margin, as a percentage, of the value axis. Only used when auto-range is true.
Category Axis Upper Margin	catAxisUpperMargin	Axes	Float8	The upper margin, as a percentage, of the category axis.
Category Axis Lower Margin	catAxisLowerMargin	Axes	Float8	The lower margin, as a percentage, of the category axis.
Chart Background	chartBackground	Appearance	Color	The backgroud color // for the chart.

Name	Scripting	Category	Property Type	Description
Plot Background	plotBackground	Appearance	Color	The backgroud color for the plot.
Series Colors	seriesColors	Appearance	Color	The sequence of colors used for series in the bar chart.
Legend?	legend	Appearance	Boolean	Should there be an item legend below the chart?
Tooltips?	tooltips	Behavior	Boolean	Should tooltips be displayed when the mouse hovers over sections?
Labels?	labels	Appearance	Boolean	Always display labels?
Gradient bars?	gradient	Appearance	Boolean	If true, bars will be painted with a gradient 'shine'.
Shadows?	shadows	Appearance	Boolean	If true, bars will have a drop-shadow beneath them.
Foreground Transparency	foregroundAlpha	Appearance	Float4	The transparency of the pie.
Category Margin	categoryMargin	Appearance	Float8	

Name	Scripting	Category	Property Type	Description
				The marign between categories as a fraction of the total space.
Item Margin	itemMargin	Appearance	Float8	The margin between bars in a category as a fraction.
Category Date Format	categoryDateFormat	Data	String	Format the category if it is a date.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

drillDown

drillDown

Is fired when drill down menu item is selected. Excludes the "Back" menu item.

Property	Description
source	The component that fired this event.
drillDownName	Text of selected drill down option menu item.
category	Value of first column for the selected row.

back

Property	Description
source	The component that fired this event.
drillDownName	Text of selected drill down option menu item.
category	Value of first column for the selected row.

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	

Property	Description
	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.

Property	Description
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	

Property	Description
	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	

Property	Description
	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

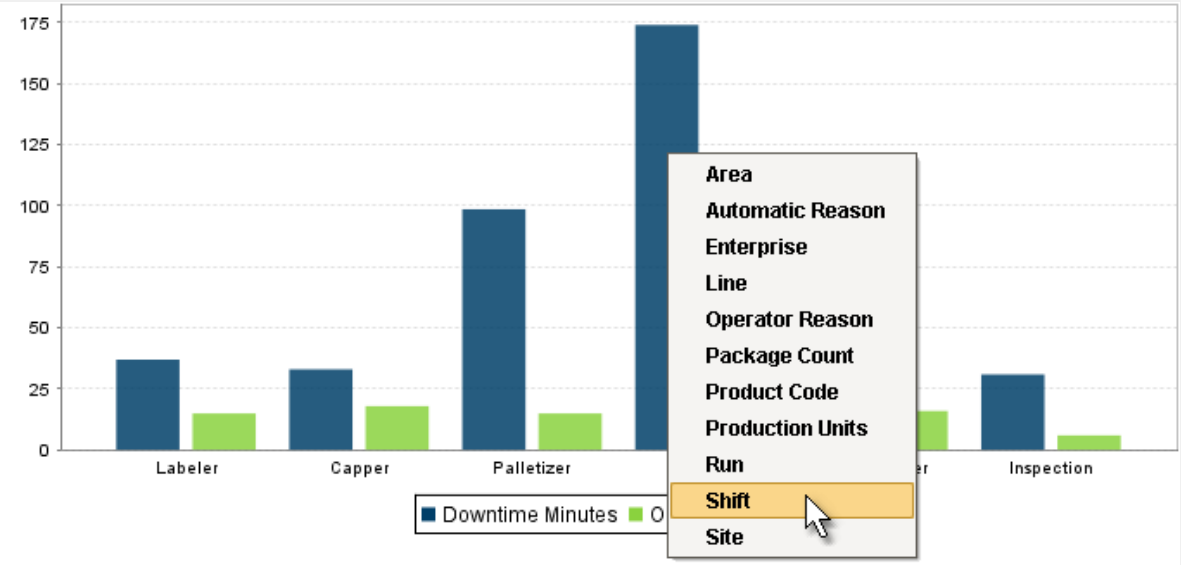
Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

This component does not have any custom properties.

Examples

When the user clicks on a bar of the bar chart, the drill down menu will appear. When an item in the drill down menu is clicked on, the drillDown event is fired. Script in the drillDown event is responsible for updating the Data property to change the results shown in the bar chart. The drill down menu information is set through the Drill Down Options property. The Drill Down Options can be populated from the Analysis Controller, Analysis Selector, SQL Query, scripting or it can be manually defined in the designer.

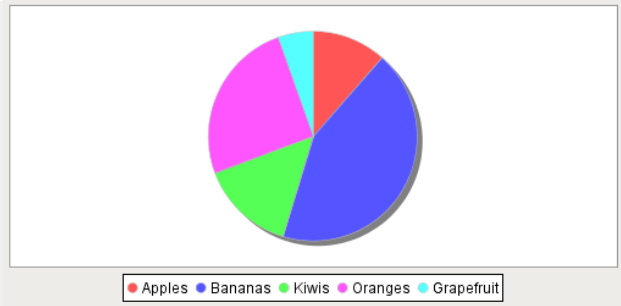


Component Bar Chart

Production Bar Chart

Production Pie Chart

General



Component Palette Icon: Production Pie Chart

Description

A component that displays a pie chart with drill down capabilities. This extends from the Pie Chart Component that comes with Ignition.

Properties

Name	Scripting	Category	Property Type	Description
Data	data	Data	Dataset	The data driving the chart.
Drill Down Options	drillDownOptions	Data	DataSet	Dataset with drill down options.
Previous Drill Down Enabled	previousDrillDownEnabled	Data	Boolean	If true, show previous in drill down menu.
Chart Title	title	Appearance	String	An optional title that will appear at the top of the chart.
	plotBackground	Appearance	Color	

Name	Scripting	Category	Property Type	Description
Plot Background				The backgroud color for all plots, unless they override it.
Section Colors	sectionColors	Appearance	Color[]	The colors to use for the pie wedge fills.
Outline Colors	outlineColors	Appearance	Color[]	The colors to use for the pie wedge outlines.
Outline Stroke	outlineStroke	Appearance	float	The width for the section outline stroke.
Legend?	legend	Appearance	boolean	Should there be an item legend below the chart?
Tooltips?	tooltips	Behavior	boolean	Should tooltips be displayed when the mouse hovers over sections?
Labels?	labels	Appearance	boolean	Should labels be displayed near sections?
Label Format	labelFormat	Appearance	String	Formatting String. '{0}' is the wedge name, '{1}' is the value, '{2}' is the percent.

Name	Scripting	Category	Property Type	Description
Tooltip Format	tooltipFormat	Appearance	String	Formatting String. '{0}' is the wedge name, '{1}' is the value, '{2}' is the percent.
Legend Font	legendFont	Appearance	Font	The font for legend items, if there is a legend.
Label Font	labelFont	Appearance	Font	The font for labels items, if there are labels.
Starting Angle	startAngle	Appearance	int	The start angle to draw the pie wedges.
Enforce Circularity?	circular	Appearance	boolean	If true, the pie cannot be an oval, even if the overall chart is.
3D?	threeDimensional	Appearance	boolean	Deprecated. Use Style property instead.
Foreground Transparency	foregroundAlpha	Appearance	double	The transparency of the pie.
3D Depth Factor	depthFactor	Appearance	Float8	The depth of a 3D pie as a factor of the chart height.

Name	Scripting	Category	Property Type	Description
Selection Highlight Color	selectionHighlightColor	Appearance	Color	The color of the selection highlight.
Selection Highlight Width	selectionHighlightWidth	Appearance	float	The line width of the selection highlight.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

drillDown

drillDown

Is fired when drill down menu item is selected. Excludes the "Back" menu item.

Property	Description
source	The component that fired this event.
drillDownName	Text of selected drill down option menu item.

Property	Description
category	Value of first column for the selected row.

back

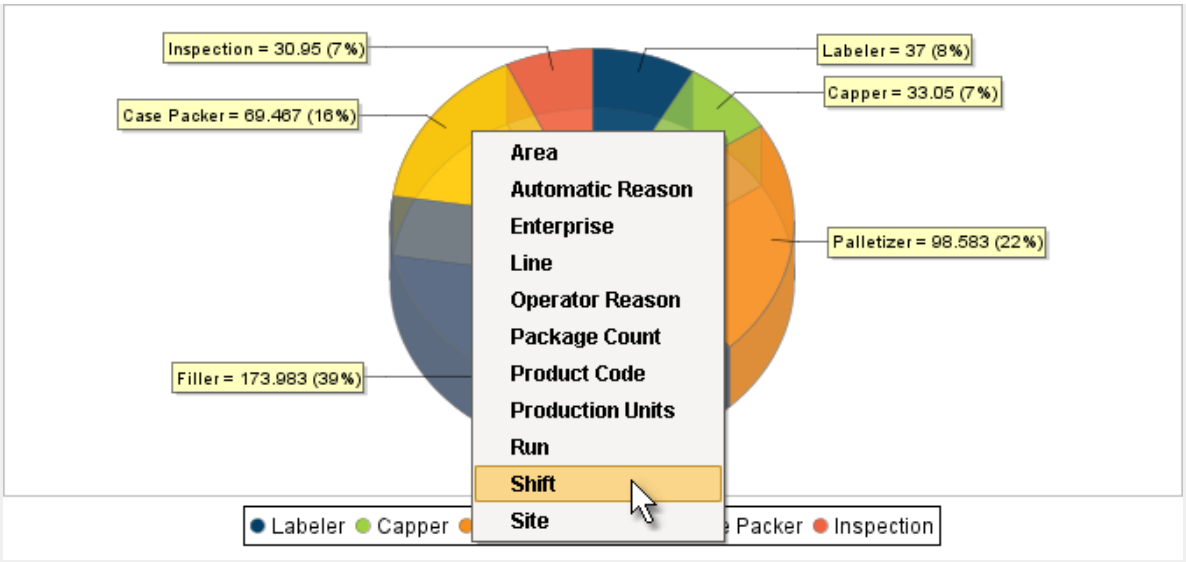
Property	Description
source	The component that fired this event.
drillDownName	Text of selected drill down option menu item.
category	Value of first column for the selected row.

Customizers

This component does not have any custom properties.

Examples

When the user clicks on a segment of the pie chart, the drill down menu will appear. When an item in the drill down menu is clicked on, the drillDown event is fired. Script in the drillDown event is responsible for updating the Data property to change the results shown in the pie chart. The drill down menu information is set through the Drill Down Options property. The Drill Down Options can be populated from the Analysis Controller, Analysis Selector, SQL Query, scripting, or it can be manually defined in the designer.



Component Pie Chart

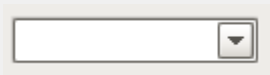
Production Pie Chart

9.5.2 Track & Trace Components

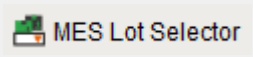
The Track and Trace module provides a set of components to provide easy user interface for track and trace data.

MES Lot Selector

General



Component Palette Icon:



Description

A component to allow auto complete selection of MES Material Lot or MES Material Sublot objects. It contains many properties to filter the MES Material Lot or MES Material Sublot objects to include in the list. The auto complete feature will include only appropriate names that start with what the user has typed. This is very useful so scrolling of large lists does not have to be done and full lot or serial number does not have to be typed in.

There are two possible modes that the MES Lot Selector component can be used. If the Mode property is set to Lot, then the selector will be populated with MES Material Lot objects. If the Mode property is set to Sublot, then the selector will be populated with MES Material Sublot objects.

The Max Results property prevents a huge number of options from being loaded from the database along with all of the overhead of passing then to the client when the user will not use all of them. This along with the Begin Date Time and End Date Time properties, keep from taking up unneeded resources.

Properties

Info

Please go through the MES Inventory Filter for more details on each property.

Name	Scripting	Category	Property Type	Description
Include Active Lots	includeActiveLots	Data	Boolean	If true, include lots tha currently being proces
Include Inactive Lots	includeInactiveLots	Data	Boolean	If true, include lots tha done being processed
Lot Status Filter	lotStatusFilter	Data	String	Filter value, including wildcard characters, to results by lot status.
	includeLotSequence	Data	Boolean	

Name	Scripting	Category	Property Type	Description
Include Lot Sequence				If true, include an item each lot number and sequence combinatio
Begin Date Time	beginDateTime	Data	DateTime	The beginning date an to include in the result valid when Include Ina Lots property is set to
End Date Time	endDateTime	Data	DateTime	The ending date and t include in the results. valid when Include Ina Lots property is set to
Lot Name Filter	lotNameFilter	Data	String	Filter value, including wildcard characters, to results by lot names.
Sublot Name Filter	sublotNameFilter	Data	String	Filter value, including wildcard characters, to results by sublot name
Lot Equipment Name Filter	lotEquipmentNameFilter	Data	String	Filter value, including wildcard characters, to results by lot location
Lot Equipment Class Name Filter	lotEquipmentClassFilter	Data	String	Filter value, including wildcard characters, to results by class of lot l names.
Personnel Name Filter	personnelNameFilter	Data	String	Filter value, including wildcard characters, to results by personnel n

Name	Scripting	Category	Property Type	Description
Personnel Class Name Filter	personnelClassFilter	Data	String	Filter value, including t wildcard characters, to results by personnel c names.
Material Name Filter	materialNameFilter	Data	String	Filter value, including t wildcard characters, to results by material nar
Material Class Name Filter	materialClassFilter	Data	String	Filter value, including t wildcard characters, to results by material clas names.
Operation Name Filter	operationNameFilter	Data	String	Filter value, including t wildcard characters, to results by operation na
Segment Name Filter	segmentNameFilter	Data	String	Filter value, including t wildcard characters, to results by segment na
Segment Equipment Name Filter	segmentEquipmentNameFilter	Data	String	Filter value, including t wildcard characters, to results by segment loc names.
Segment Equipment Class Name Filter	segmentEquipmentClassFilter	Data	String	Filter value, including t wildcard characters, to results by class of seg location names.
Custom Property Value Filter	customPropertyValueFilter	Data	String	List of custom property and value to filter resu (E.g. "MaterialLot. CustomPropertyName

Name	Scripting	Category	Property Type	Description
Selected UUID	selectedUUID	Data	String	The selected operatio
Selected Name	selectedName	Data	String	The selected operatio
Selected Lot UUID	selectedLotUUID	Data	String	The lot UUID for the c selected lot.
Selected Lot Name	selectedLotName	Data	String	The lot Name for the c selected lot.
Selected Sublot UUID	selectedSublotUUID	Data	String	The sublot UUID for th currently selected lot.
Selected Sublot Name	selectedSublotName	Data	String	The sublot Name for t currently selected lot.
Partial Results	partialResults	Data	Boolean	The selected operatio

Scripting

Scripting Functions

refresh

- Description

Updates contents to reflect any updates to the database since the object was created, or since the last refresh. By default, refresh is automatic for local operations when view navigation touches an update.

- Parameters

None

- Return

Nothing

- Scope

Client

Extension Functions

lotSelected

- Description

Called before an MES Material is selected. Return false to prevent the MES Material Lot from being started.

- Parameters

self - A reference to the component that is invoking this function

mesMaterialLot - The MESMaterialLot reference object. Use mesMaterialLot.getMESObject() to get the MES Material Lot itself

- Return

1

- Scope

Client

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.

Property	Description
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.

Property	Description
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.

Property	Description
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	

Property	Description
	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	

Property	Description
	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

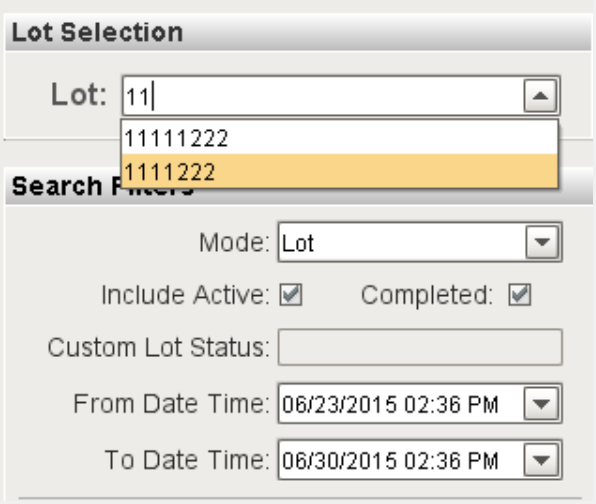
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

This component does not have any custom properties.

Examples

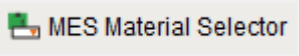


Property Name	Value
Begin Date Time	06/23/2015 02:41 PM
End Date Time	06/30/2015 02:41 PM
Include Active Lots	True

MES Material Selector

General

Component Palette Icon:



Description

A component that is added to Ignition windows to display material selection components for the active MES Response Segment. The material selection components that are shown depend on the configuration of the MES Operations Segment.

In the image below, a MES Operations Segment with a material resource named Raw Material caused the Raw Material header. The material resource has a material class defined causing the component to select the specific material to be added.

The location for the material is not known, so a component to select the location is added and will contain options based on the Equipment Class to store the material at.

The material resource is also configured for manual lot number source, causing a component to accept the lot number is added.

And last, the material source is configured for manual entry of the quantity causing a component to accept quantity is added.

All this is performed automatically and there is no need to create custom Ignition windows for each combination of how MES Operations Segment objects are configured.

Properties

Name	Scripting	Category	Property Type	Description
Name Filter	nameFilter	Data	String	Filter lot names.
	modifiedIconPath	Appearance	String	

Name	Scripting	Category	Property Type	Description
Modified Icon Path				The relative path of an icon image to indicate modified material values.
Alert Icon Path	alertIconPath	Appearance	String	The relative path of an icon image to indicate alert material values.
Max Quantity Decimal Digits	quantityMaxDecimalDigits	Data	Int4	The maximum number of decimal digits in the quantity input. Default = 0.
Max Lot Return Count	maxLotReturnCount	Data	Int4	The maximum number of lots to return from equipment location. Default=2, Range 2-6.
Lot Name Pattern	lotNamePattern	Data	String	The lot name pattern as a combination of LotNumber, EquipmentName, MaterialName separated by commas. Default=LotNumber, EquipmentName.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

itemSelected

- Description

Called after an MES lot or material is selected. In this function, lot can be changed on the fly when a selection is made. Return False to ignore the change.

- Parameters

self - A reference to the component that is invoking this function

mesObjectLink - MESObjectLink object containing the MESMaterialLot or MESMaterialDef details. Call mesObjectLink.getMESObject() to get the instance of the MESMaterialLot or MESMatrialDef object.

- Return

1

- Scope

Client

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	

Property	Description
	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

Property	Description
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

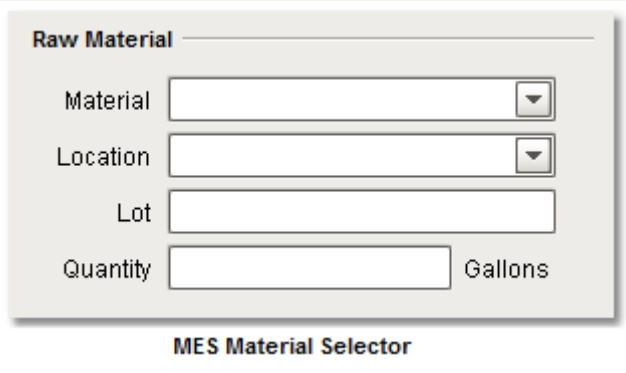
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

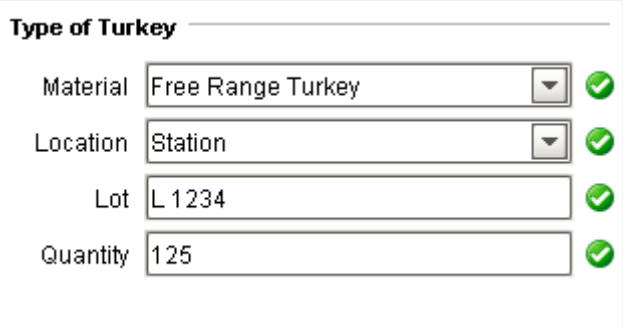
Customizers

This component does not have any custom properties.

Examples



MES Material Selector



Property Name	Value
Max Quantity Decimal Digits	0
Max Lot Return Count	2
Lot Name Pattern	LotNumber
Modified Icon Path	Builtin/icons/16/check2.png

If more than one material resource has been defined for a MES Operations Segment, then the MES Material Selector component will automatically populate components for each material resource.

The MES Material Selector must be used in conjunction with the MES Segment Selector component. The active MES Response Segment is retrieved from the MES Segment Selector component. MES Operations Response objects are derived from MES Operations Segment objects and drive the entry components created in the MES Material Selector component. No binding is required for the two to work together. Behind the scenes, the MES Material Selector finds the MES Segment Selector and the two will communicate.

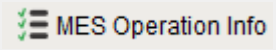
Info

For the MES Material Selector component to find the MES Segment Selector, it must be in the same container on the window. It is okay to be in a container, they just both have to be in the same container or root container. Multiple containers can exist on the same window containing separate MES Segment Selector and MES Material Selector components in each. The components residing in the same container will work together allowing multiple segments to be controlled from the same window.

MES Operation Info

General

Component Palette Icon:



Description

A component to display details of an MES Operation Response. In addition to using this component to display the details, script can also be used. Properties of the MES Operation Info component control what details to show. By setting the Show Material Info property to True, will cause all material details to be shown. The same is true for operations, segment, material sublot, personnel and custom properties.

Properties

Name	Scripting	Category	Property Type	Descrip
Equipment Path	equipmentPath	Data	String	The equipme path wit the producti model t show availabl operatic
Mode	mode	Data	Integer	The sele for Real or Histo data dis (0 or 1 respecti
Operation Response Link	mesObjectLink	Data	MESObjectLink	Bind to operatic respons to show details f
Show Operation Info	showOperationInfo	Data	Boolean	If true, s informat for the M operatic
Show Equipment Info	showEquipmentInfo	Data	Boolean	If true, s process equipme informat
Show Segment Info	showSegmentInfo	Data	Boolean	

Name	Scripting	Category	Property Type	Descrip
				If true, s informat for the a MES segmen
Show Supplemental Equipment Info	showSupplementalEquipmentInfo	Data	Boolean	If true, s supplem equipme informat
Show Material Info	showMaterialInfo	Data	Boolean	If true, s MES material informat
Show Material Sublot Info	showSublotInfo	Data	Boolean	If true, s MES material sublot informat
Show Personnel Info	showPersonnelInfo	Data	Boolean	If true, s MES personn informat
Show Custom Property Info	showCustomPropertyInfo	Data	Boolean	If true, s associat custom properti
Section Title Font	sectionTitleFont	Appearance	Font	Font to for secti titles.
Section Label Font	sectionLabelFont	Appearance	Font	

Name	Scripting	Category	Property Type	Descrip
				Font to for secti lebels.
Item Label Font	itemLabelFont	Appearance	Font	Font to for item labels.
Item Value Font	itemValueFont	Appearance	Font	Font to for item values.
Date Format	dateFormat	Behavior	String	The dat formatti pattern to forma string versions the date

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	

Property	Description
	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.

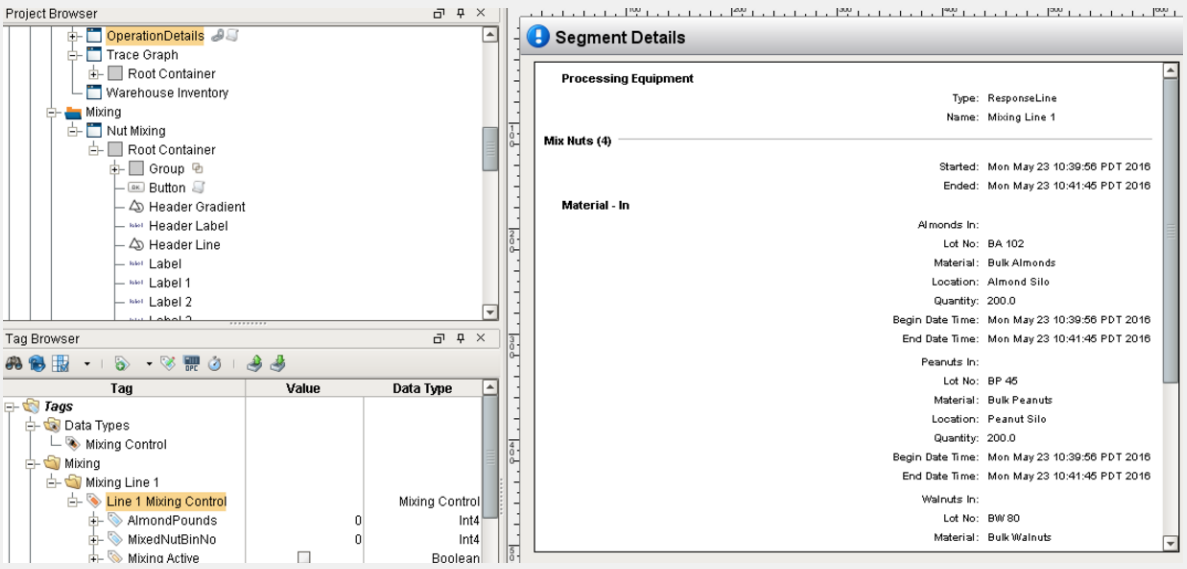
Property	Description
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

This component does not have any custom properties.

Example

The example below shows details about the materials involved in a mixing operation and also the equipment details.

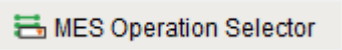


Property Name	Value
Show Equipment Info	True
Show Material Info	True

MES Operation Selector

General

Component Palette Icon:



Description

A component that is added to Ignition windows to display and select operations definitions. The operations definitions that are shown, are limited by those appropriate for the equipment specified in the equipment path property. The auto complete feature will include only appropriate operations definitions that start with what the user has typed. This is very useful so scrolling of large lists does not have to be done and the full name of the operations definition does not have to be typed in.

Properties

Name	Scripting	Category	Property Type	Description
Equipment Path	equipmentPath	Data	String	The equipment path within the production model to show available operations.

Selected UUID	selectedUUID	Data	String	The selected operation UUID.
Selected Name	selectedName	Data	String	The selected operation Name.
Active	active	Data	Boolean	True if the selected operation is active.
Can Begin Operation	canBeginOperation	Data	Boolean	True if the selected operation can be started.
Can End Operation	canEndOperation	Data	Boolean	True if the selected operation is started and can be ended.
Name Filter	nameFilter	Data	String	Filter operation names.

Scripting

Scripting Functions

beginOperation

- Description

This script function invokes the extension function 'beginOperation'.

- Parameters

None

- Return

Nothing

- Scope

Client

endOperation

- Description

This script function invokes the extension function 'endOperation'.

- Parameters

None

- Return

Nothing

- Scope

Client

abortOperation

- Description

Aborts the operation.

- Parameters

None

- Return

Nothing

- Scope

Client

Extension Functions

beginOperation

- Description

Called before an MES Operation begins. Return false to prevent the MES Operation from being started.

- Parameters

self - A reference to the component that is invoking this function

mesOperationResponse - The MESOperationResponse object itself. Core and custom properties can be set on the object before the operation begins.

- Return

```
1
```

- Scope

Client

endOperation

- Description

Called before an MES Operation ends. Return false to prevent the MES Operation from being ended.

- Parameters

self - A reference to the component that is invoking this function.

mesOperationResponse - The MESOperationResponse object itself. Core and custom properties can be set on the object before the operation ends.

- Return

1

- Scope

Client

operationSelected

- Description

Called after an MES Operation is selected. In this function, operations can be started. This allows for operation to automatically start when the user selects a MES Operation.

- Parameters

self - A reference to the component that is invoking this function

mesObjectLink - MESObjectLink object containing the MES Operation details. call mesObjectLink.getMESObject() to get the instance of the MESOperationDefinition object.

- Return

1

- Scope

Client

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.

Property	Description
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.

Property	Description
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.

Property	Description
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.

Property	Description
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.

Property	Description
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

This component does not have any custom properties.

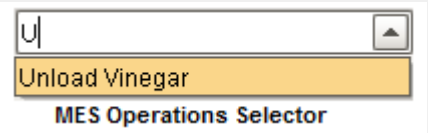
Example

The Track and Trace module provides a set of components to enter in data for each operation. In fact the demo project comes with quite a few screens that already does this.

Open the "Trace" project in the designer and open the "Unload Vinegar" window from the "Main Windows" folder. You can now specify the operators to select an operation for. Click on the "Operation" dropdown list. Set the "Equipment Path" property to:

My Enterprise\Site1\Raw Materials\Unload Station

Go into preview mode and dropdown the list. Then fix the "Start" and "Stop" buttons on the window.



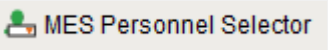
Property Name	Value
Active	True
Can Begin Operation	True

Property Name	Value
Can End Operation	True
Equipment Path	My Enterprise\Site1\Raw Materials\Unload Station

MES Personnel Selector

General

Component Palette Icon:



Descripton

A component that is added to Ignition windows to display personnel selection components for the active MES Response Segment. The personnel selection components that are shown depend on the configuration of the MES Operations Segment.

In the image below, a MES Operations Segment with a personnel resource named Operator caused the Operator header. The personnel resource has a personnel class defined causing the component to select the specific person to be added.

All this is performed automatically and there is no need to create custom Ignition windows for each combination of how MES Operations Segment objects are configured.

Properties

Name	Scripting	Category	Property Type	Description
Name Filter	nameFilter	Data	String	Filter lot names.
Modified Icon Path	modifiedIconPath	Appearance	String	The relative path of an icon image to indicate modified personnel values.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

itemSelected

- Description

Called after an MES person is selected. In this function, person can be changed on the fly when a selection is made. Return False to ignore the change.

- Parameters

self - A reference to the component that is invoking this function

mesObjectLink - MESObjectLink object containing the MES Person details. Call mesObjectLink.getMESObject() to get the instance of the MESPerson object.

- Return

1

- Scope

Client

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	

Property	Description
	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

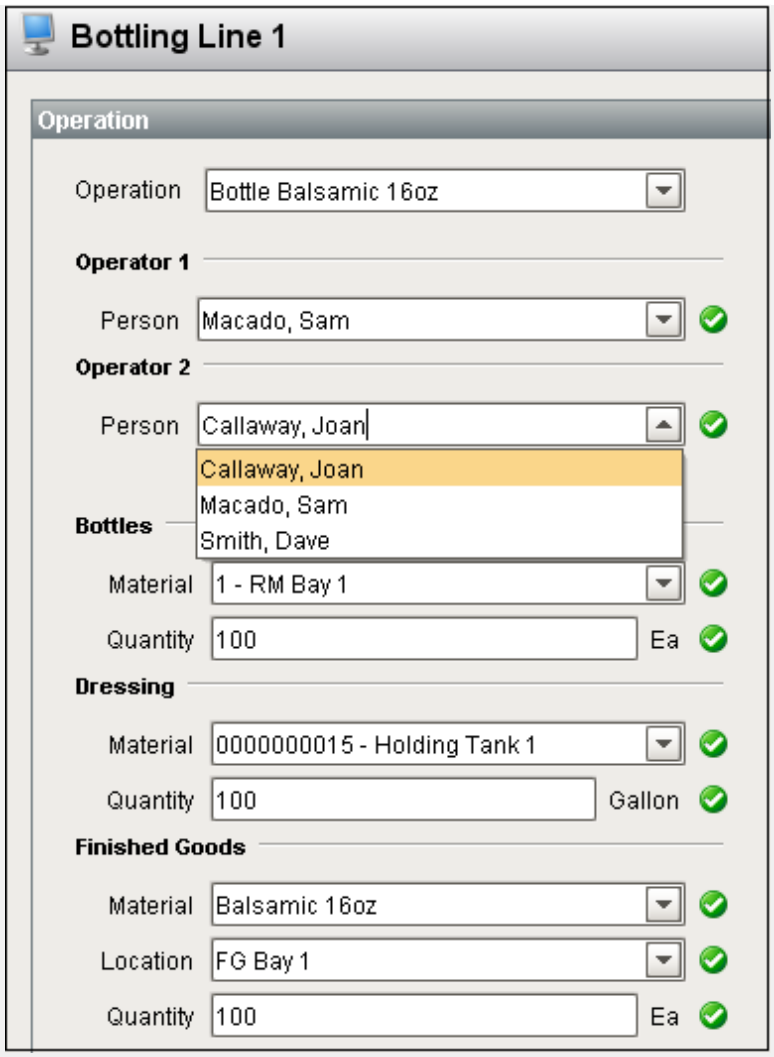
Property	Description
source	The component that fired this event.

Property	Description
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

This component does not have any custom properties.

Examples



Property Name	Value
Modified Icon Path	Builtin/icons/16/check2.png

If more than one personnel resource has been defined for a MES Operations Segment, then the MES Personnel Selector component will automatically populate components for each personnel resource.

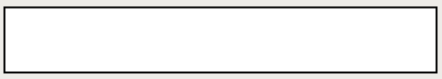
The MES Personnel Selector must be used in conjunction with the MES Segment Selector component. The active MES Response Segment is retrieved from the MES Segment Selector component. MES Operations Response objects are derived from MES Operations Segment objects and drive the entry components created in the MES Personnel Selector component. No binding is required for the two to work together. Behind the scenes, the MES Personnel Selector finds the MES Segment Selector and the two will communicate.

Info

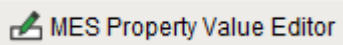
For the MES Personnel Selector component to find the MES Segment Selector, it must be in the same container on the window. It is okay to be in a container, they just both have to be in the same container or root container. Multiple containers can exist on the same window containing separate MES Segment Selector and MES Personnel Selector components in each. The components residing in the same container will work together allowing multiple segments to be controlled from the same window.

MES Property Value Editor

General



Component Palette Icon:



Description

A component to display MES property value components for the active MES Response Segment. The MES property value entry components that are shown depend on the configuration of the MES associated objects.

In the image below, a MES Operations Segment with a material resource that references a MES Material Definition that has a Viscosity custom property. This is causing the entry component to accept a viscosity value be added. In order for the custom property to be shown, the Production Visible option for the custom property must be set to True. Likewise, the required indication will be shown when the Required option for the custom property is set to True.

All this is performed automatically and there is no need to create custom Ignition windows for each combination of how associated MES objects are configured.

Properties

Name	Scripting	Category	Property Type	Description
Name Filter	nameFilter	Data	String	Filter lot names.
Required Icon Path	requiredIconPath	Appearance	String	The relative path of an icon image to indicate required property values.
Modified Icon Path	modifiedIconPath	Appearance	String	The relative path of an icon image to indicate modified property values.
Read Only	readOnly	Data	Boolean	Allow editing of editor.
Mode	mode	Data	int	Select mode to work with a segment selector or sublot list component.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	

Property	Description
	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	**Description**
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

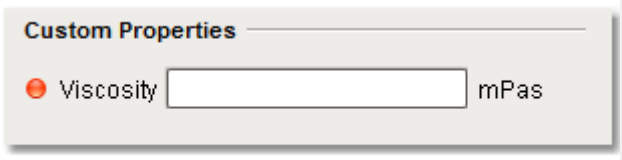
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

This component does not have any custom properties.

Examples



MES Property Value Editor

Property Name	Value
Read Only	False

If more than one property has been defined for the associated MES objects, then the MES Property Value Editor component will automatically populate components for each one.

The MES Property Value Editor must be used in conjunction with the MES Segment Selector or MES Sublot List components. Which one it works with is determined by the Mode property of this component.

If the Mode is set to Segment, then the active MES Response Segment is retrieved from the MES Segment Selector component. MES Response Segment objects are derived from MES Operations Segment objects and drive the entry components created in the MES Property Value Editor component. No binding is required for the two to work together. Behind the scenes, the MES Property Value Editor finds the MES Segment Selector and the two will communicate.

If the Mode is set to Lot, then the active MES Response Segment is retrieved from the MES Segment Selector component. All lot references will be scanned for MES properties and drive the entry components created in the MES Property Value Editor component. No binding is required for the two to work together. Behind the scenes, the MES Property Value Editor finds the MES Segment Selector and the two will communicate.

If the Mode property is set to Sublot, then the selected MES Material Sublot is retrieved from the MES Sublot List component. The MES Material Sublot object drive the entry components created in the MES Property Value Editor component. No binding is required for the two to work together. Behind the scenes, the MES Property Value Editor finds the MES Sublot List and the two will communicate.

Info

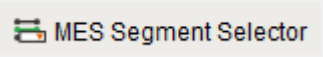
For the MES Property Value Editor component to find the MES Segment Selector or MES Sublot List, it must be in the same container on the window. It is okay to be in a container, they just both have to be in the same container or root container. Multiple containers can exist on the same window containing separate MES Segment Selector and/or MES Sublot List and MES Property Value Editor components in each. The components residing in the same container will work together allowing multiple segments to be controlled from the same window.

If two MES Property Value Editor components exist on the same container and one is set to Segment mode and the other is set the Sublot mode, the two will connect to the correct parent. This allows MES property values to be entered for both the segment or lot and for the selected sublot on the same screen.

MES Segment Selector

General

Component Palette Icon:



Description

A component that is added to Ignition windows to display and select operations segments. The operations segments that are shown, are limited by those appropriate for the equipment specified in the equipment path property. The auto complete feature will include only appropriate operations definitions that start with what the user has typed. This is very useful so scrolling of large lists does not have to be done and the full name of the operations segment does not have to be typed in.

The MES Segment Selector component has two different modes.

Segment Mode

The Segment mode is used to select a Operations Segment to run based on an active Operations Response for the specified equipment. This is needed when there are multiple segments being used for an operation.

Definition Mode

If all that is needed is to run a single segment like Unload Vinegar then, the segment can automatically be selected when the operation is selected. The Definition mode enables this functionality and the MES Operations Selector can be left off the screen. In order for the MES Segment Selector to know which segment to select, one of two situations must exist. Either, there must only be one Operations Segment in the Operation Definition or the Operations Segment name must match the Operations Definition name.

Scripting

Properties

Name	Scripting	Category	Property Type	Description
Equipment Path	equipmentPath	Data	String	The equipment path within the production model to show available segments.
Auto End Operation	autoEndOperation	Data	Boolean	When true, automatically end the operation when the segment is ended.
Active	active	Data	Boolean	True if the selected segment is active.
Can Begin Segment	canBeginSegment	Data	Boolean	True if the selected segment can be started.
Can End Segment	canEndSegment	Data	Boolean	True if the selected segment is started and can be ended.
Can Update Segment	canUpdateSegment	Data	Boolean	True if the selected segment is started and and has been modified and can be updated.
Can Undo Segment	canUndoSegment	Data	Boolean	True if changes to the selected segment can be undone.
Selected UUID	selectedUUID	Data	String	The selected segment UUID.
	selectedName	Data	String	

Name	Scripting	Category	Property Type	Description
Selected Name				The selected segment Name.
Name Filter	nameFilter	Data	String	Filter operation names.
Mode	mode	Data	String	Determines whether the segment or both the segment and the Operation are selected

Scripting Functions

beginSegment

- Description

Invokes the beginSegment extension function.

- Parameters

None

- Return

Nothing

- Scope

Client

updateSegment

- Description

Invokes the updateSegment extension function.

- Parameters

None

- Return

Nothing

- Scope

Client

executeSegment

- Description

Executes the segment.

- Parameters

None

- Return

Nothing

- Scope

Client

endSegment

- Description

Invokes the endSegment extension function.

- Parameters

None

- Return

Nothing

- Scope

Client

endAllSegments

- Description

Invokes the endSegment extension function for each segments.

- Parameters

None

- Return

Nothing

- Scope

Client

undoChanges

- Description

Undo the changes made to the segment.

- Parameters

None

- Return

Nothing

- Scope

Client

Extension Functions

beginSegment

- Description

Begin the specified response segment.

- Parameters

String responseSegment - The MESResponse segment object to begin. All required property values must be set prior to beginning.

- Return

Nothing

- Scope

Client

updateSegment

- Description

Update an active segment. If material, personnel, supplemental equipment resources or custom properties change during a production task, then the update script function is used to commit the changes.

- Parameters

String responseSegment - The MESResponse segment object to be updated.

- Return

Nothing

- Scope

Client

endSegment

- Description

End the specified response segment.

- Parameters

String responseSegment - The MESResponse segment object to end. All final property values must be set prior to ending.

- Return

Nothing

- Scope

Client

segmentSelected

- Description

Called after an MES Segment is selected. In this function, segments can be started provided the required lot, material, equipment and personnel properties have been set. This allows for operation to automatically start when the user selects a MES Segment.

- Parameters

String mesObjectLink - MES object containing the MES Segment details. Call mesObjectLink.getMESObject() to get the instance of the MESOperationSegment object.

- Return

Nothing

- Scope

Client

beginOperation

- Description

Called before an MES Operation begins. Return false to prevent the MES Operation from being started.

- Parameters

mesOperationResponse - The MESOperationResponse object itself. Core and custom properties can be set on the object before the operation begins.

- Return

True

- Scope

Client

endOperation

- Description

Called before an MES Operation ends. Return false to prevent the MES Operation from being ended.

- Parameters

mesOperationResponse - The MESOperationResponse object itself. Core and custom properties can be set on the object before the operation ends.

- Return

True

- Scope

Client

optionsUpdated

- Description

Called when the available options have been updated and currently there are no active segments.

- Parameters

mesObjectList - A list of MESObjectLink objects representing the segments that will appear in the drop down list.

- Return

Return the index of the option to select, otherwise return None to not select any.

- Scope

Client

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	

Property	Description
	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

Property	Description
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

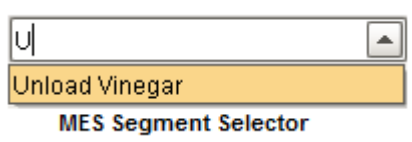
Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	

Property	Description
	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

This component does not have any custom properties.

Examples

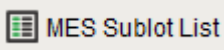


Property Name	Value
Equipment Path	[global]\My Enterprise\California\Receiving\Unload Station 1
Auto End Operation	True

MES Sublot List

General

Component Palette Icon:



Description

A component that is added to Ignition windows to manage material sublots. The user can view, add, edit or delete sublot items. MES Material Sublots are used to track individual items that belong to an MES Material Lot. For example, if each item that is part of a lot has a serial number, then this component can be used to allow the operator to manage them. It can also be done in script.

Properties

Name	Scripting	Category	Property Type	Description
Show Item Number	showItemNumber	Appearance	Boolean	If true, show the sequential item number.
Row Height	rowHeight	Appearance	int	The row hight of a list entry.
Modified Icon Path	modifiedIconPath	Appearance	String	The relative path of an icon image to indicate modified property values.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	**Description**
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	

Property	Description
	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

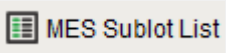
Property	Description
source	The component that fired this event.

Property	Description
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

This component does not have any custom properties.

Examples



Property Name	Value
Modified Icon Path	Builtin/icons/16/check2.png
Show Item Number	False
Row Height	24

The MES Sublot List must be used in conjunction with the MES Segment Selector component. The active MES Response Segment is retrieved from the MES Segment Selector component. MES Operations Response objects are derived from MES Operations Segment objects and drive the MES Material Sublot objects to show. If the MES Operations Segment object has multiple material resources that have the Enable Sublots

setting set to True, then a selection component will appear in this component allowing the operation to select the lot first. If only one material resource has the Enable Sublots setting set to True, then the component will not be added and just the sublot list will appear.

No binding is required for the two to work together. Behind the scenes, the MES Material Selector finds the MES Segment Selector and the two will communicate.

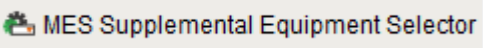
Info

For the MES Sublot List component to find the MES Segment Selector, it must be in the same container on the window. It is okay to be in a container, they just both have to be in the same container or root container. Multiple containers can exist on the same window containing separate MES Segment Selector and MES Sublot List components in each. The components residing in the same container will work together allowing multiple segments to be controlled from the same window.

MES Supplemental Equipment Selector

General

Component Palette Icon:



Description

A component that is added to Ignition windows to select or specify MES Supplemental Equipment using auto complete.

Properties

Name	Scripting	Category	Property Type	Description
Name Filter	nameFilter	Data	String	Filter lot names.
Modified Icon Path	modifiedIconPath	Appearance	String	The relative path of an icon image to indicate modified supplemental equipment values.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

itemSelected

- Description

Called after a supplemental equipment item is selected. In this function, equipment can be changed on the fly when a selection is made. Return False to ignore the change.

- Parameters

self - A reference to the component that is invoking this function

mesObjectLink - MESObjectLink object containing the MES Equipment details. Call mesObjectLink.getMESObject() to get the instance of the MESEquipment object.

- Return

1

- Scope

Client

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	

Property	Description
	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

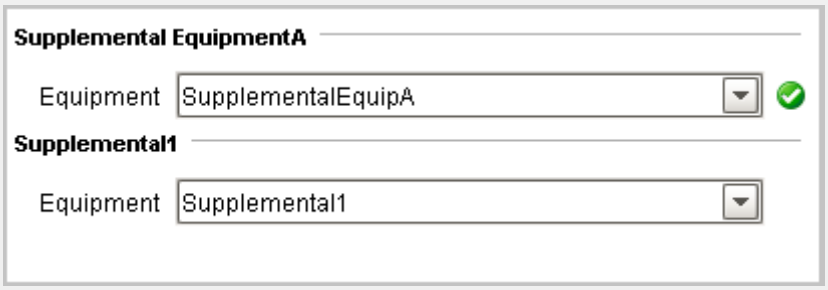
Property	Description
source	The component that fired this event.

Property	Description
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

This component does not have any custom properties.

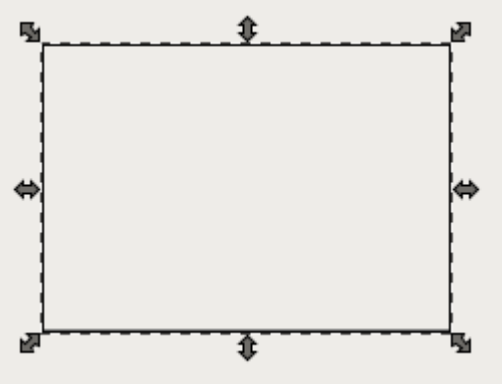
Examples



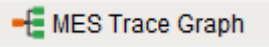
Property Name	Value
Modified Icon Path	Builtin/icons/16/check2.png

MES Trace Graph

General



Component Palette Icon:



Description

A component that is added to Ignition windows to visually see traceability results. It shows the flow of production for bulk lot (batch) and / or serialized items. This allows entering a lot (batch) number and seeing what went into making it up from raw materials through the production steps to the finished goods. Then if desired, product can be tracked beyond the production facility. Individual items can also tracked by using a serial number or other item identification.

Properties

Name	Scripting	Category	Property Type	Description
Lot Name	lotName	Behavior	String	Lot name to sho graph for. If this will ignore the Lo property.
Lot UUID	lotUUID	Behavior	String	Lot UUID to sho graph for. If the L Name property is this property will ignored.

Name	Scripting	Category	Property Type	Description
Highlight Sublot Name	highlightSublotName	Behavior	String	Optionally, set th of the sub lot to highlight.
Lot Tool Tip Format	lotToolTipFormat	Appearance	String	Set to format the of lot node toolti formatted in HTM can reference th column names ir dataset that the graph display is generated from. dataset results a same as the Lot Binding Function Sublot Trace Bin Function that has documentation o available columr Example<html> {LotName} {LotSequence}<b {LotStatus} {LotBeginDateTi />{LotEndDateTi />{MaterialName {LotLocationNam /html> Possible Tooltips LotUUID LotName SublotUUID SublotName LotSequence LotUse LotBeginDateTir

Name	Scripting	Category	Property Type	Description
				LotEndDateTime LotQuantity LotStatus MaterialUUID MaterialName LotLocationUUID LotLocationNam SegmentUUID SegmentName SegmentBeginD SegmentEndDat SegmentLocatio SegmentLocatio PrevSegmentUU PrevLotUUID NextSegmentUU NextLotUUID SegInCount SegOutCount LotInCount LotOutCount LotContainsSubl LotAvailability LotDescription LotEnabled LotAssembly LotUnits MaterialDescript MaterialEnabled EquipmentUUID EquipmentName EquipmentDescr EquipmentPath EquipmentEnabl

Name	Scripting	Category	Property Type	Description
				SublotDescriptio SublotEnabled SublotAssembly SublotStatus Sublots CustomPropertie
Segment Tool Tip Format	segmentToolTipFormat	Appearance	String	Set to format to the segment noc tooltip. It is forma HTML and can reference the co names in the dat that the trace gra display is genera from. The datase results are the sa the Lot Trace Bi Function and Su Trace Binding Fu that has full documentation c available column Example:<html> {SegmentName} {SegmentLocatic </html> Possible Tooltips LotUUID LotName SublotUUID SublotName LotSequence LotUse LotBeginDateTir LotEndDateTime

Name	Scripting	Category	Property Type	Description
				LotQuantity LotStatus MaterialUUID MaterialName LotLocationUUI LotLocationNam SegmentUUID SegmentName SegmentBeginD SegmentEndDat SegmentLocatio SegmentLocatio PrevSegmentUU PrevLotUUID NextSegmentUU NextLotUUID SegInCount SegOutCount LotInCount LotOutCount LotContainsSubl LotAvailability LotDescription LotEnabled LotAssembly LotUnits MaterialDescript MaterialEnabled EquipmentUUID EquipmentName EquipmentDescr EquipmentPath EquipmentEnabl SublotDescriptio SublotEnabled SublotAssembly SublotStatus

Name	Scripting	Category	Property Type	Description
				Sublots CustomPropertie
User Menu Items	userMenuItems	Behavior	DataSet	A dataset that st user menu items
Enable Auto Sizing	enableAutoSizing	Behavior	Boolean	If true, auto size mouse clicked o space.
Node Configuration	nodeConfiguration	Behavior	DataSet	A dataset that st node configurati
Edge Color	edgeColor	Appearance	Color	The color of the between nodes.
Sublot Match Text	sublotMatchText	Appearance	String	Text to display w lot contains the specified sublot
Breadcrumbs Visible	breadcrumbVisible	Appearance	Boolean	Displays a list of breadcrumb links previously viewe information.
Breadcrumb Font	breadcrumbFont	Appearance	Font	The font used to the breadcrumb
Breadcrumb Color	breadcrumbColor	Appearance	Color	The color of the breadcrumb links
Breadcrumb Underlined	breadcrumbUnderlined	Appearance	Boolean	Show the breadc links as underlin
Breadcrumb Max Count	breadcrumbMax	Appearance	int	The maximum c breadcrumbs to history.

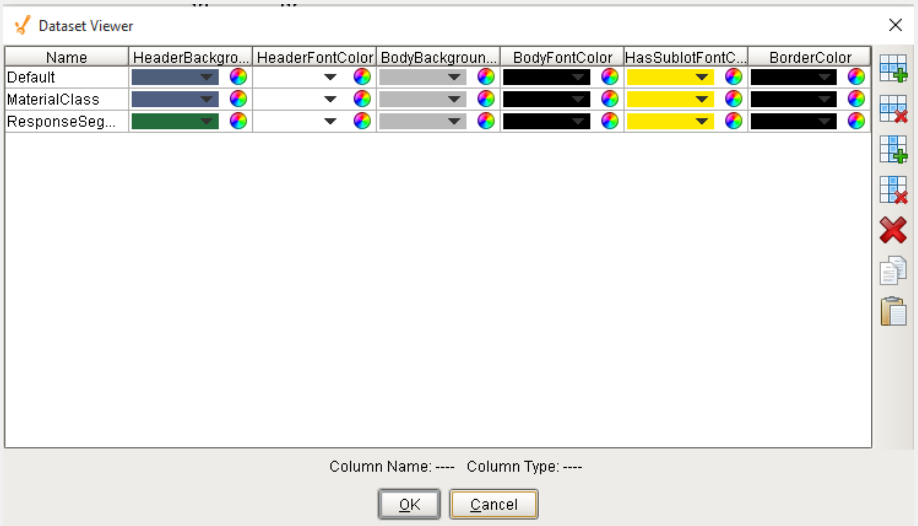
Name	Scripting	Category	Property Type	Description
Breadcrumb Icon Path	breadcrumbSeparatorIconPath	Appearance	String	The relative path icon image used separate the breadcrumb link
Editor Title Font	editorTitleFont	Appearance	Font	Font for titles of panels.
Category Font	categoryFont	Appearance	Font	Font for category in editing table.
Property Font	propertyFont	Appearance	Font	Font for properti editing table.
Description Area Font	descriptionAreaFont	Appearance	Font	Font for descript area of editing ta
Button Font	buttonFont	Appearance	Font	Font for buttons.
Close Button Font	closeButtonFont	Appearance	Font	The font of the b close the editing
Miscellaneous Font	miscellaneousFont	Appearance	Font	Font for miscella components.
Title Background Color	titleBackgroundColor	Appearance	Color	The color of the background in th editing panel.
Title Text Color	titleTextColor	Appearance	Color	The color of the in the editing par
Close Button Color	closeButtonColor	Appearance	Color	The color of the to close the editi panel.

Name	Scripting	Category	Property Type	Description
Category Background Color	categoryBackgroundColor	Appearance	Color	The background category rows.

Node Configuration Property

This property controls the appearance of each node. Click on the **Dataset Viewer** icon for the **Behaviour** property in **Property Editor** to set the values.

Name will filter the MES object name that should be included as nodes. Default, MaterialClass and ResponseSegment are the values inbuilt on Ignition, as shown below. HeaderBackground is the color of the node header. HeaderFontColor will decide the color of the text. Color of the node is determined by BodyBackground. BodyFontColor will decide the color of the text. The color of the nodes with sublots will be controlled by HasSublotFontColor. BorderColor is the color of the border.



Scripting

Scripting Functions

autoFit

- Description

The trace graph is automatically resized as necessary to fit. Zooms a display such that all items within a given group will fit within the display bounds. By default, this achieved by clicking the right mouse button once, with no dragging.

- Parameters

None

- Return

Nothing

- Scope

Client

Example

```
event.source.parent.getComponent('MES Trace Graph').
autoFit()
```

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

menu

userMenuItemClicked

This event fires when the menu item is clicked, or if the user selects the menu item using the keyboard and presses the Enter key. It can also occur if an access key or shortcut key is pressed that is associated with the MenuItem.

Property	Description
source	The component that fired this event.
menuItemName	Name of the user menu item that triggered the event.

Property	Description
nodeName	Name of the node. This is the same as the name of the MES object that is associated with the node.
objectType	Name of the MES object type that is associated with the node.
uuid	UUID of the MES object that is associated to the node.
lotUUID	UUID of the material lot.
lotName	Name of the material lot.
lotSequence	The sequence number associated with the material lot.
lotUse	The lot use type of the material.
beginDateTime	Date and Time at which the event was triggered.
materialUUID	UUID of the material.
materialName	Name of the material.
lotEquipmentUUID	UUID of the equipment lot.
lotEquipmentName	Name of the equipment lot.
segmentUUID	UUID of the segment.
segmentName	Name of the segment.
segmentEquipmentUUID	UUID of the segment equipment.
segmentEquipmentName	Name of the segment equipment.

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

Property	Description
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

Property	Description
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	

Property	Description
	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

selectNode

nodeClicked

The NodeClick event is generated when the user clicks a particular Node object.

Property	Description
source	The component that fired this event.
nodeName	Name of the node. This is the same as the name of the MES object that is associated with the node.
objectType	Name of the MES object type that is associated with the node.
uuid	UUID of the MES object that is associated to the node.
lotUUID	UUID of the material lot.
lotName	Name of the material lot.
lotSequence	The sequence number associated with the material lot.
lotUse	The lot use type of the material.
beginDateTime	Date and Time to begin the segment.
endDateTime	Date and Time to end the segment.
materialUUID	UUID of the material.
materialName	Name of the material.
lotEquipmentUUID	UUID of the equipment lot.

lotEquipmentName	Name of the equipment lot.
segmentUUID	UUID of the segment.
segmentName	Name of the segment.
segmentEquipmentUUID	UUID of the segment equipment.
segmentEquipmentName	Name of the segment equipment.

nodeEntered

This event is generated when the Node is being hovered over.

Property	**Description**
source	The component that fired this event.
nodeName	Name of the node. This is the same as the name of the MES object that is associated with the node.
objectType	Name of the MES object type that is associated with the node.
uuid	UUID of the MES object that is associated to the node.
lotUUID	UUID of the material lot.
lotName	Name of the material lot.
lotSequence	The sequence number associated with the material lot.
lotUse	The lot use type of the material.
beginDateTime	Date and Time at which the event was triggered.
endDateTime	Date and Time to end the segment.

Property	Description
materialUUID	UUID of the material.
materialName	Name of the material.
lotEquipmentUUID	UUID of the equipment lot.
lotEquipmentName	Name of the equipment lot.
segmentUUID	UUID of the segment.
segmentName	Name of the segment.
segmentEquipmentUUID	UUID of the segment equipment.
segmentEquipmentName	Name of the segment equipment.

nodeExited

This event is generated when drag gesture exits this `Node` .

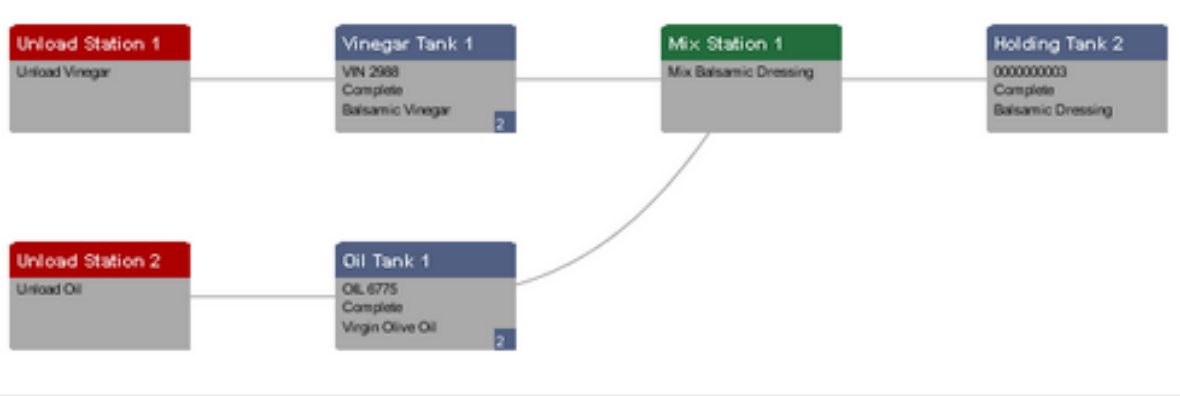
Property	Description
source	The component that fired this event.
nodeName	Name of the node. This is the same as the name of the MES object that is associated with the node.
objectType	Name of the MES object type that is associated with the node.
uuid	UUID of the MES object that is associated to the node.
lotUUID	UUID of the material lot.
lotName	Name of the material lot.
lotSequence	The sequence number associated with the material lot.

Property	Description
lotUse	The lot use type of the material.
beginDateTime	Date and Time at which the event was triggered.
endDateTime	Date and Time to end the segment.
materialUUID	UUID of the material.
materialName	Name of the material.
lotEquipmentUUID	UUID of the equipment lot.
lotEquipmentName	Name of the equipment lot.
segmentUUID	UUID of the segment.
segmentName	Name of the segment.
segmentEquipmentUUID	UUID of the segment equipment.
segmentEquipmentName	Name of the segment equipment.

Customizers

This component does not have any custom properties.

Examples

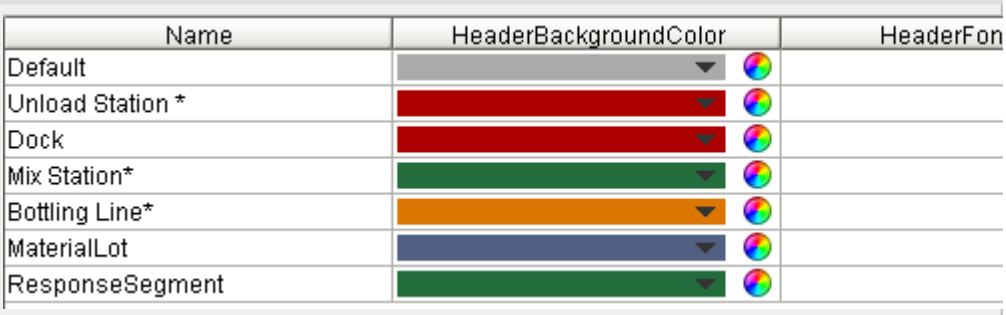


MES Trace Graph

Property Name	Value
Enable_Auto_Sizing	True
Breadcrumbs_Visible	True
Breadcrumb_Underlined	True
Breadcrumb_Max_Count	20
Lot_Name	VN 2988
Lot_Tool_Tip_Format	Possible Tooltips are: LotUUID LotName SublotUUID SublotName LotSequence LotUse LotBeginDateTime LotEndDateTime LotQuantity

Property Name	Value
	LotStatus MaterialUUID MaterialName LotLocationUUID LotLocationName SegmentUUID SegmentName SegmentBeginDateTime SegmentEndDateTime SegmentLocationUUID SegmentLocationName PrevSegmentUUID PrevLotUUID NextSegmentUUID NextLotUUID SegInCount SegOutCount LotInCount LotOutCount LotContainsSublot LotAvailability LotDescription LotEnabled LotAssembly LotUnits MaterialDescription MaterialEnabled EquipmentUUID EquipmentName EquipmentDescription EquipmentPath EquipmentEnabled SublotDescription SublotEnabled SublotAssembly SublotStatus

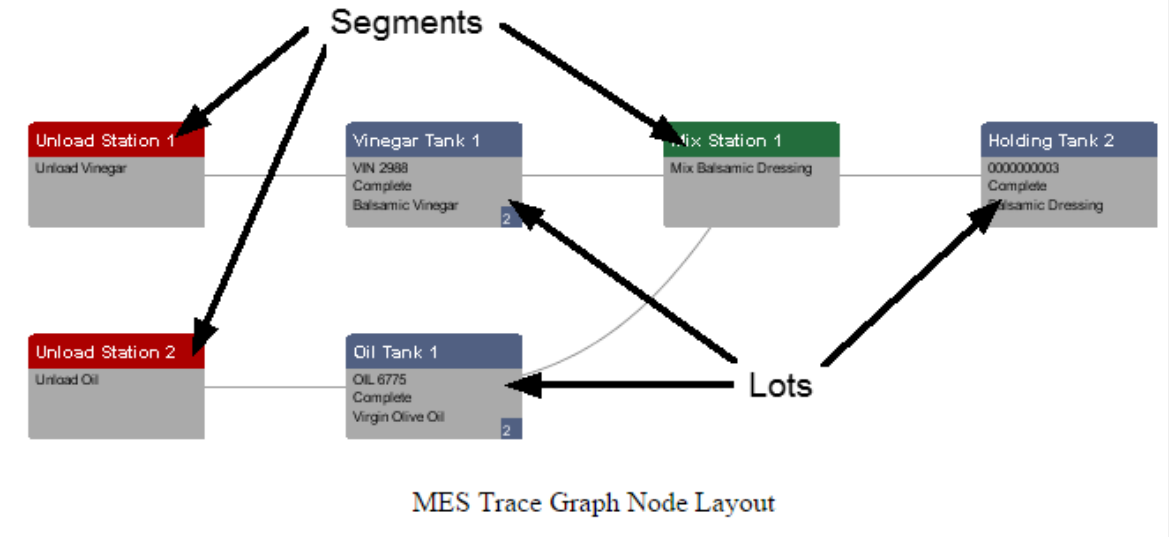
Property Name	Value
	Sublots CustomProperties
Segment_ Tool_ Tip_ Format	Possible Tooltips are: LotUUID LotName SublotUUID SublotName LotSequence LotUse LotBeginDateTime LotEndDateTime LotQuantity LotStatus MaterialUUID MaterialName LotLocationUUID LotLocationName SegmentUUID SegmentName SegmentBeginDateTime SegmentEndDateTime SegmentLocationUUID SegmentLocationName PrevSegmentUUID PrevLotUUID NextSegmentUUID NextLotUUID SegInCount SegOutCount LotInCount LotOutCount LotContainsSublot LotAvailability LotDescription LotEnabled LotAssembly

Property Name	Value
	LotUnits MaterialDescription MaterialEnabled EquipmentUUID EquipmentName EquipmentDescription EquipmentPath EquipmentEnabled SublotDescription SublotEnabled SublotAssembly SublotStatus Sublots CustomProperties
Sublot_Match_Text	Sublot Match
Node_Configuration	

If a node has a sublot that has a name that matches the Highlight Sublot Name, then text specified by the Sublot Match Text property will be displayed in the lot node. This is valuable at determining which lots contain the sublot (serial number) of interest.

The nodes are laid out in chronological order from left to right. The node type alternates starting with a segment then showing a lot. The idea behind this is there are lots that are inputs to an operation and there are lot that the operation produced. In the image below, the upper left node titled Unload Station 1 is the operation that vinegar was unloaded.

When this operation was done, a new lot VIN 2988 was created. Then that lot was used in the operation of making of balsamic dressing at Mix Station 1, which produced balsamic dressing that resides in Holding Tank 2.



MES Trace Graph Node Layout

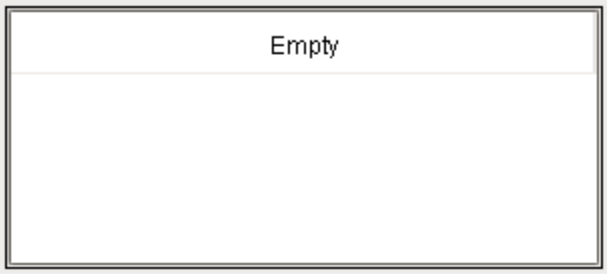
The Trace Graph component also is an excellent navigation tool to zero in on non-trace information. Because the date and times, material, equipment, lot numbers, serial number, etc. are known, other data can be filtered to match the trace information being shown. This gives, otherwise just time series data, context to specific lots and serialized items without the need to look it up manually potentially across multiple systems.

To support this functionality, the trace graph component has very configurable menus that are used to display additional non-trace information. When the menu is selected by the user, the associated date and times, lot number, material, personnel, etc. is included in the menu event so that data within and outside of Ignition can be looked up and displayed.

9.5.3 OEE Components

OEE Down Time Table

General



Component Palette Icon:  OEE Down Time Table

In this Page

Description

A component that displays OEE downtime events for an active production run and allows the operator to select more specific downtime reasons for the event. It also allows the operator to split downtime events. This accommodates downtime events that have multiple reasons. For example, if a production line goes down because of a mechanical failure and when maintenance finishes the repair, it is time for break. The operator can split the downtime event into two events. One for mechanical failure and the other for break.

Properties

Name	Scripting	Category	Property Type	Description
Equipment Path	equipmentPath	Data	String	The equipment path to show or edit downtime reasons for. If the equipment path points to the Line, all events that are considered to cause line downtime are shown. If the equipment path points a cell under a line, all

Name	Scripting	Category	Property Type	Description
				downtime events for that cell will be shown.
Downtime Reason List View Type	downtimeReasonListViewType	Behavior	int	The type of the downtime reason list to show. Choices are 0 (Grid) and 1 (Tree).
Excluded Equipment Path	excludedEquipmentPath	Data	String	The beginning part of a path to exclude from the displayed equipment paths.
Run Look Back Count	runLookBackCount	Data	Integer	The number of runs to show downtime events for within the selected date range. Set to 0 to see all runs within the date range. Set to 1 to see only the current run. Greater numbers are additive: 2 will show you events

Name	Scripting	Category	Property Type	Description
				from the last 2 runs, 3 from the last 3 runs, et cetera. ⚠ If set to 1 or greater, the date range will not be used.
Rollup Time Span	rollupTimeSpan	Data	Integer	The rollup time span in seconds to combine events by.
Start Date	startDate	Data	Date	The start date to get downtime reasons for the equipment.
End Date	endDate	Data	Date	The end date to get downtime reasons for the equipment.
Editable	editable	Data	Boolean	Determines whether downtime reasons are edited.

Name	Scripting	Category	Property Type	Description
Enable Notes	enableNotes	Data	Boolean	Determines whether notes are shown or edited for downtime reasons.
Activity Timeout	activityTimeout	Data	Integer	Number of seconds to wait after user activity before update.
Column Attribute Data	columnAttributesData	Data	Dataset	The dataset containing the data attributes
Data	data	Data	Dataset	The data for this table.
Selected Row	selectedRow	Data	int	The index of the first selected row, or -1 if none.
Selection Foreground Color	selectionForeground	Appearance	Color	The foreground color of a selected row in the table.
Selection Background Color	selectionBackground	Appearance	Color	The background color of a selected row in the table.
	verticalScrollbarWidth	Appearance	int	

Name	Scripting	Category	Property Type	Description
Vertical Scrollbar Width				The width of a vertical scrollbar.
Horizontal Scrollbar Height	horizontalScrollbarHeight	Appearance	int	The height of a horizontal scrollbar.
Header Font	headerFont	Appearance	Font	The font of text of the table header.
Header Foreground Color	headerForeground	Appearance	Color	The foreground color of the table header.
Grid Line Color	gridColor	Appearance	Color	The color of grid lines in the table.
Show Horizontal Grid Lines?	showHorizontalLines	Appearance	Boolean	Determines whether horizontal grid lines are shown in the table.
Show Vertical Grid Lines?	showVerticalLines	Appearance	Boolean	Determines whether vertical grid lines are shown in the table.
Title Font	titleFont	Appearance	Font	The font of text of the title bar.
	titleForeground	Appearance	Color	

Name	Scripting	Category	Property Type	Description
Title Foreground Color				The foreground color of the title bar.
Title Background Color	titleBackground	Appearance	Color	The background color of the title bar.
Slide Font	slideFont	Appearance	Font	The font of text of the slide panel.
Slide Foreground Color	slideForeground	Appearance	Color	The foreground color of the slide panel.
Slide Background Color	slideBackground	Appearance	Color	The background color of the slide panel.
Slide Type	slideType	Appearance	int	The type of the slide panel to open it. Options: Over, Out.
Slide Direction	slideDirection	Appearance	int	The direction of the slide panel to open it. Options: Left, Right, Top, Bottom.
Maximum Slide Position	maximumSlidePosition	Appearance	float	The maximum position of the slide panel to open it.

Name	Scripting	Category	Property Type	Description
Auto Row Height Enabled	autoRowHeightEnabled	Appearance	Boolean	If true, the row height of the downtime table will be adjusted automatically.
Row Height	rowHeight	Appearance	int	The row height of the downtime table
Downtime Reason Button Width	downtimeReasonButtonWidth	Appearance	int	The width of downtime reason buttons in the downtime reason grid view.
Downtime Reason Button Height	downtimeReasonButtonHeight	Appearance	int	The height of downtime reason buttons in the downtime reason grid view.
Equipment State Class Icon Path	equipmentStateClassIconPath	Appearance	String	The relative path of the 'Equipment State Class' icon image of the downtime reason list view. The recommended icon size is 16x16 pixels.
	equipmentStateIconPath	Appearance	String	

Name	Scripting	Category	Property Type	Description
Equipment State Icon Path				The relative path of the 'Equipment State' icon image of the downtime reason list view. The recommended icon size is 16x16 pixels.
Change Equipment Icon Path	changeEquipmentIconPath	Appearance	String	The relative path of the 'Change Equipment' icon image. The recommended icon size is 16x16 pixels.
Revert to Original Code Icon Path	revertToOriginalCodeIconPath	Appearance	String	The relative path of the 'Revert to Original Code' icon image. The recommended icon size is 16x16 pixels.
Split Downtime Reason Icon Path	splitDowntimeReasonIconPath	Appearance	String	The relative path of the 'Split Downtime Reason' icon image. The recommended icon size is 16x16 pixels.

Name	Scripting	Category	Property Type	Description
Note Downtime Reason Icon Path	noteDowntimeReasonIconPath	Appearance	String	The relative path of the 'Note Downtime Reason' icon image. The recommended icon size is 16x16 pixels.
Slider Knob Icon Path	sliderKnobIconPath	Appearance	String	The relative path of the slider knob icon image in the Split Downtime Reason view.
Left Arrow Icon Path	leftArrowIconPath	Appearance	String	The relative path of the left arrow icon image in the Split Downtime Reason view.
Right Arrow Icon Path	rightArrowIconPath	Appearance	String	The relative path of the right arrow icon image in the Split Downtime Reason view.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

configureCell

- Description

Provides a chance to configure the contents of each cell.

- Parameters

self - A reference to the component that is invoking this function.

value - The value in the dataset at this cell.

textValue - The text the table expects to display at this cell (may be overridden by including 'text' attribute in returned dictionary).

selected - A boolean indicating whether this cell is currently selected.

rowIndex - The index of the row in the underlying dataset.

colIndex - The index of the column in the underlying dataset.

colName - The name of the column in the underlying dataset.

rowView - The index of the row, as it appears in the table view (affected by sorting).

colView - The index of the column, as it appears in the table view (affected by column re-arranging and hiding).

- Returns

Return a dictionary of name-value pairs with the desired attributes. Available attributes include: 'background', 'border', 'font', 'foreground', 'horizontalAlignment', 'iconPath', 'text', 'toolTipText', 'verticalAlignment'

You may also specify the attribute 'renderer', which is expected to be a javax.swing.JComponent which will be used to render the cell.

- Scope

Client

onColumnsCreate

- Description

Called when columns are created in the table. Provides a chance to add custom columns to the table.

- Parameters

self - A reference to the component that is invoking this function.

- Returns

Returns a dictionary of custom column name-type pairs.

- Scope

Client

onRowAdd

- Description

Called when a row is added in the table. Provides a chance to insert values to custom columns in the table.

- Parameters

reason - Down time reason of a row.

code - Down time code of a row.

- Returns

Returns a dictionary of custom column name-value pairs.

- Scope

Client

configureHeaderStyle

- Description

Provides a chance to configure the style of each columnn header. Return a dictionary of name-value pairs with the desired attributes. Available attributes include: 'background', 'border', 'font', 'foreground', 'horizontalAlignment', 'toolTipText', 'verticalAlignment'.

- Parameters

self - A reference to the component that is invoking this fucntion.

colIndex - The index of the column in the underlying dataset.

colName - The name of the column in the underlying dataset.

- Returns

Returns a dictionary of name-value pairs with the desired attributes.

- Scope

Client

initialize

- Description

Called when the window containing this table is opened, or the template containing it is loaded. Provides a chance to initialize the table further, for example, setting the default row configuration.

- Parameters

self - A reference to the component that is invoking this fucntion.

- Returns

Nothing

- Scope

Client

loadIcon

- Description

Provides a chance to change an icon. Based on the icon name parameter, return the image path to the icon to use in place of the default icon.

- Parameters

self - A reference to the component that is invoking this function.

iconName - The name of the icon.

- Returns

Nothing

- Scope

Client

Example

```
# This example will return a path to a different image to
replace the default delate image:
if iconName == 'remove':/n/t/treturn 'Builtin/icons/24
/delete2.png'
```

Event Handlers

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

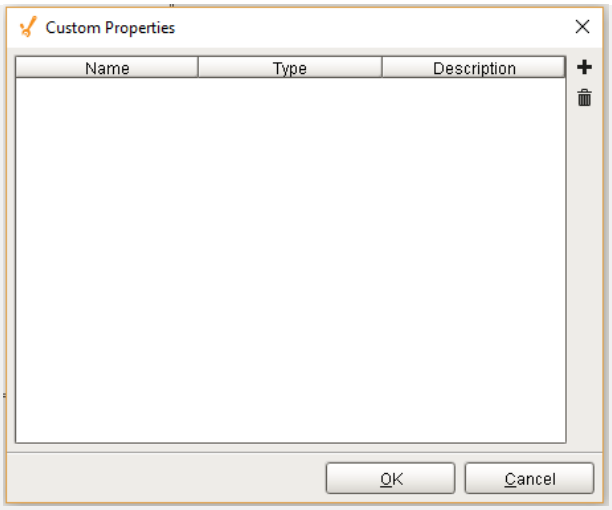
Table Customizer shown below manages the data entered into the Down Time Table.

Using the Table Customizer

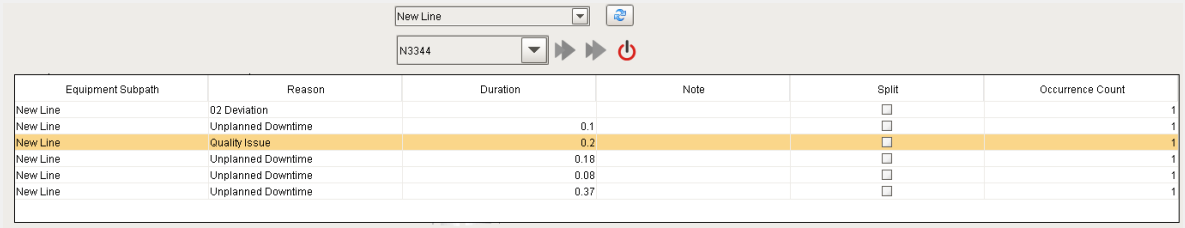
A table customizer is available by right clicking the down time table in the designer and selecting "Customizers" -> "Table Customizer". It is similar to the table customizer in a standard Ignition table.

Custom Properties

The custom properties can be used to add user defined properties.



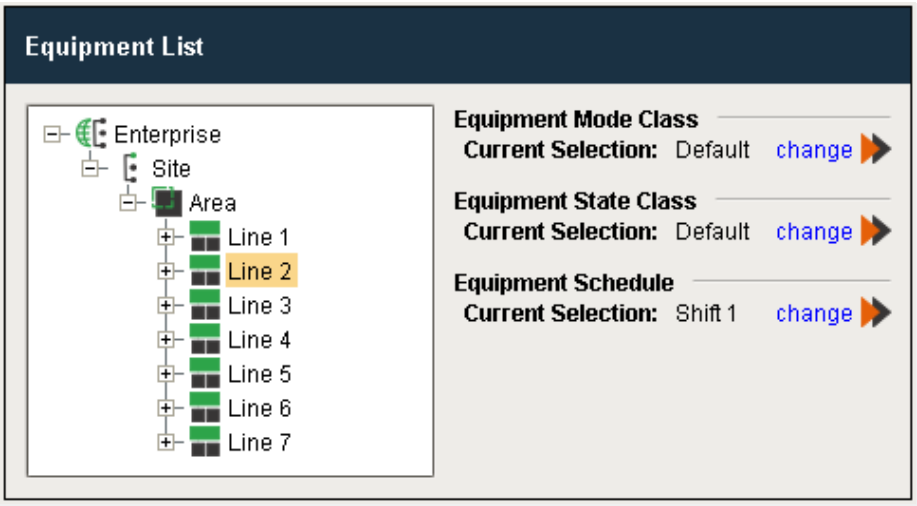
Examples



Property Name	Value
Equipment Path	Nuts Unlimited\Folsom\Packaging\New Line

OEE Equipment Manager

General



Component Palette Icon:

OEE Equipment Manager

In this Page

See Setting Up Equipment Modes page and Setting Up Equipment States page for various OEE Equipment Manager settings.

Description

OEE Equipment Manager component is used to modify MES equipment states, modes and schedules. The change button change will navigate to the window where you can add, edit, delete, copy, paste, import and export the mode, state and schedule.

Properties

Name	Scripting	Category	Property Type	Description

Name	Scripting	Category	Property Type	Description
Enable Mode Editing	enableModeEditing	Behavior	Boolean	If true, the Equipment Mode is editable.
Enable State Editing	enableStateEditing	Behavior	Boolean	If true, the Equipment State is editable.
Enable Schedule Editing	enableScheduleEditing	Behavior	Boolean	If true, the Equipment Schedule is editable.
Equipment Path Filter	equipmentPathFilter	Data	String	The equipment path filter, that can include * and ? wildcard characters, to filter the equipments by.
Title Font	titleFont	Appearance	Font	The font of text of the title bar.
Title Foreground Color	titleForeground	Appearance	Color	The foreground color of the title bar.
	titleBackground	Appearance	Color	

Name	Scripting	Category	Property Type	Description
Title Background Color				The background color of the title bar.
Equipment Mode Root Icon Path	equipmentModeRootIconPath	Appearance	String	The relative path of the 'Equipment Mode Root' icon image of the equipment mode tree view. The recommended icon size is 16x16 pixels.
Equipment Mode Class Icon Path	equipmentModeClassIconPath	Appearance	String	The relative path of the 'Equipment Mode Class' icon image of the equipment mode tree view. The recommended icon size is 16x16 pixels.
Equipment Mode Icon Path	equipmentModeIconPath	Appearance	String	The relative path of the 'Equipment Mode' icon image of the equipment mode tree view. The

Name	Scripting	Category	Property Type	Description
				recommended icon size is 16x16 pixels.
New Equipment Mode Class Icon Path	newEquipmentModeClassIconPath	Appearance	String	The relative path of the 'New Equipment Mode Class' icon image. The recommended icon size is 16x16 pixels.
New Equipment Mode Icon Path	newEquipmentModeIconPath	Appearance	String	The relative path of the 'New Equipment Mode' icon image. The recommended icon size is 16x16 pixels.
Equipment State Root Icon Path	equipmentStateRootIconPath	Appearance	String	The relative path of the 'Equipment State Root' icon image of the equipment state tree view. The recommended icon size is 16x16 pixels.

Name	Scripting	Category	Property Type	Description
Equipment State Class Icon Path	equipmentStateClassIconPath	Appearance	String	The relative path of the 'Equipment State Class' icon image of the equipment state tree view. The recommended icon size is 16x16 pixels.
Equipment State Icon Path	equipmentStateIconPath	Appearance	String	The relative path of the 'Equipment State' icon image of the equipment state tree view. The recommended icon size is 16x16 pixels.
New Equipment State Class Icon Path	newEquipmentStateClassIconPath	Appearance	String	The relative path of the 'New Equipment State Class' icon image. The recommended icon size is 16x16 pixels.
	newEquipmentStateIconPath	Appearance	String	

Name	Scripting	Category	Property Type	Description
New Equipment State Icon Path				The relative path of the 'New Equipment State' icon image. The recommended icon size is 16x16 pixels.
Edit Icon Path	editIconPath	Appearance	String	The relative path of the 'Edit' icon image. The recommended icon size is 16x16 pixels.
Delete Icon Path	deleteIconPath	Appearance	String	The relative path of the 'Delete' icon image. The recommended icon size is 16x16 pixels.
Copy Icon Path	copyIconPath	Appearance	String	The relative path of the 'Copy' icon image. The recommended icon size is 16x16 pixels.
Paste Icon Path	pasteIconPath	Appearance	String	The relative path of the 'Paste' icon

Name	Scripting	Category	Property Type	Description
				image. The recommended icon size is 16x16 pixels.
Import Icon Path	importIconPath	Appearance	String	The relative path of the 'Import' icon image. The recommended icon size is 16x16 pixels.
Export Icon Path	exportIconPath	Appearance	String	The relative path of the 'Export' icon image. The recommended icon size is 16x16 pixels.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

loadIcon

- Description

Provides a chance to change an icon. Based on the icon name parameter, return the image path to the icon to use in place of the default icon.

- Parameters

self - A reference to the component that is invoking this function.

iconName - The name of the icon.

- Returns

Nothing

- Scope

Client

Example

```
# This example will return a path to a different image to
replace the default delate image:
if iconName == 'remove':/n/t/treturn 'Builtin/icons/24
/delete2.png'
```

Event Handlers

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

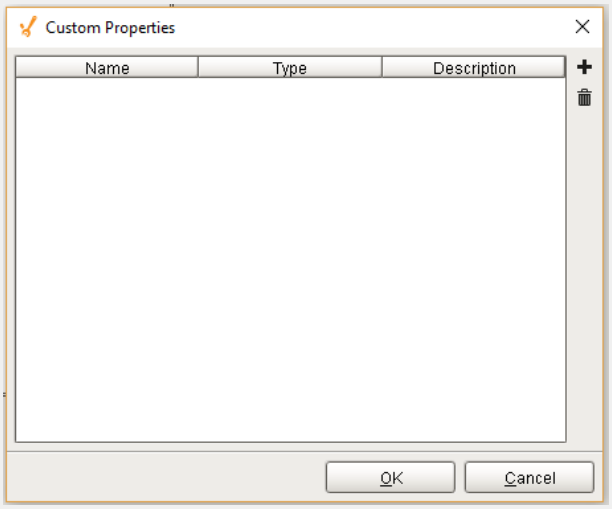
Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	

Property	Description
	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

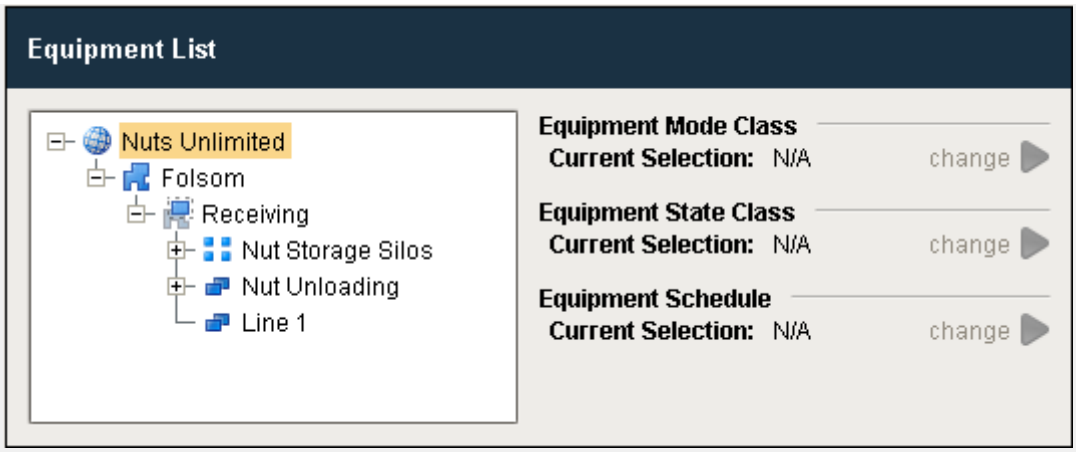
Customizers

Custom Properties

The custom properties can be used to add user defined properties.

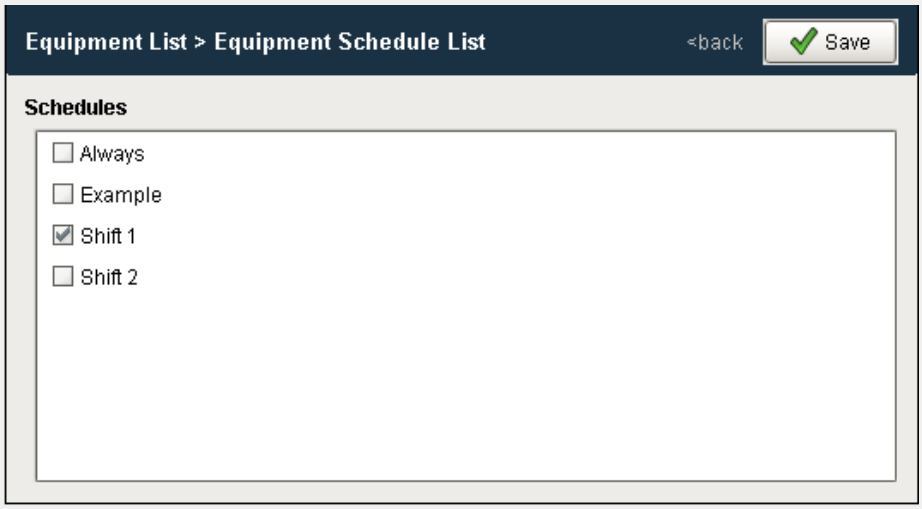


Example

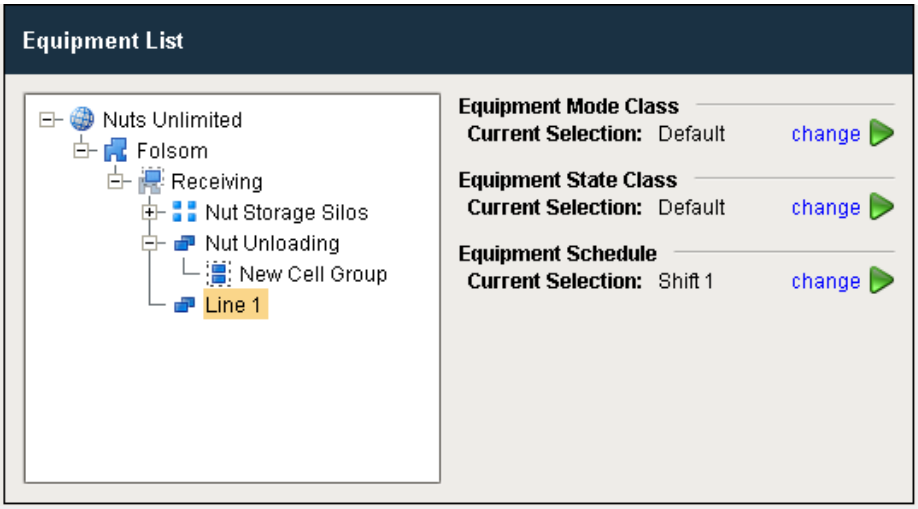


Property Name	Value
Equipment Path Filter	Nuts Unlimited\Folsom\Receiving*

To change the schedule, select the production item (for example Line 1) and then click the change icon. The equipment schedule list appears. Now select the desired shift (in this example Shift 1) and save the settings.

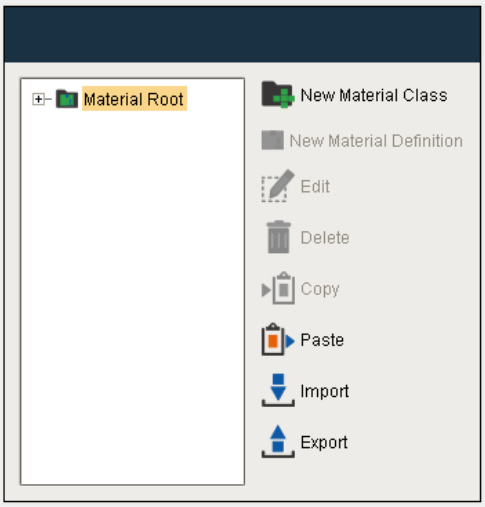


The equipment manager will now display the current selection as Shift 1.

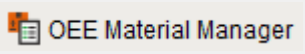


OEE Material Manager

General



Component Palette Icon:



In this Page

See Creating Materials page for various OEE Material Manager settings.

Description

OEE Material Manager component to modify or configure MES materials for OEE. The 

is the folder that holds all the material classes. Expand the material classes to see the material definitions. This component can also be used to add, edit, delete, copy, paste, import and export a material definition or the material class.

Properties

Name	Scripting	Category	Property Type	Description
Title Font	titleFont	Appearance	Font	The font of text of the title bar.
Title Foreground Color	titleForeground	Appearance	Color	The foreground color of the title bar.
Title Background Color	titleBackground	Appearance	Color	The background color of the title bar.
Material Root Icon Path	materialRootIconPath	Appearance	String	The relative path of the 'Material Root' icon image of the material tree view. The recommended icon size is 16x16 pixels.

Name	Scripting	Category	Property Type	Description
Material Class Icon Path	materialClassIconPath	Appearance	String	The relative path of the 'Material Class' icon image of the material tree view. The recommended icon size is 16x16 pixels.
Material Definition Icon Path	materialDefinitionIconPath	Appearance	String	The relative path of the 'Material Definition' icon image of the material tree view. The recommended icon size is 16x16 pixels.
New Material Class Icon Path	newMaterialClassIconPath	Appearance	String	The relative path of the 'New Material Class' icon image. The recommended icon size is 16x16 pixels.
New Material Definition Icon Path	newMaterialDefinitionIconPath	Appearance	String	The relative path of the 'New Material Definition' icon

Name	Scripting	Category	Property Type	Description
				image. The recommended icon size is 16x16 pixels.
Material Name Filter	materialNameFilter	Data	String	The material name filter, that can include * and ? wildcard characters, to filter the materials by.
Edit Icon Path	editIconPath	Appearance	String	The relative path of the 'Edit' icon image. The recommended icon size is 16x16 pixels.
Delete Icon Path	deleteIconPath	Appearance	String	The relative path of the 'Delete' icon image. The recommended icon size is 16x16 pixels.
Copy Icon Path	copyIconPath	Appearance	String	The relative path of the 'Copy' icon image. The recommended icon size is 16x16 pixels.

Name	Scripting	Category	Property Type	Description
Paste Icon Path	pasteIconPath	Appearance	String	The relative path of the 'Paste' icon image. The recommended icon size is 16x16 pixels.
Import Icon Path	importIconPath	Appearance	String	The relative path of the 'Import' icon image. The recommended icon size is 16x16 pixels.
Export Icon Path	exportIconPath	Appearance	String	The relative path of the 'Export' icon image. The recommended icon size is 16x16 pixels.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

loadIcon

- Description

Provides a chance to change an icon. Based on the icon name parameter, return the image path to the icon to use in place of the default icon.

- Parameters

self - A reference to the component that is invoking this function.

iconName - The name of the icon.

- Returns

Nothing

- Scope

Client

Example

```
# This example will return a path to a different image to
replace the default delate image:
if iconName == 'remove':/n/t/treturn 'Builtin/icons/24
/delete2.png'
```

Event Handlers

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

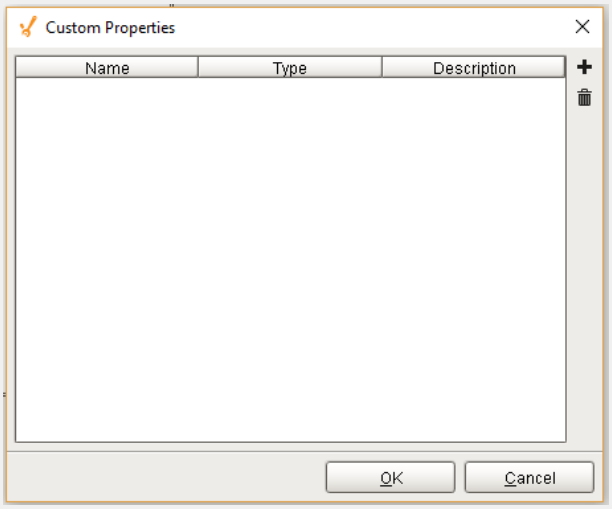
Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	

Property	Description
	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

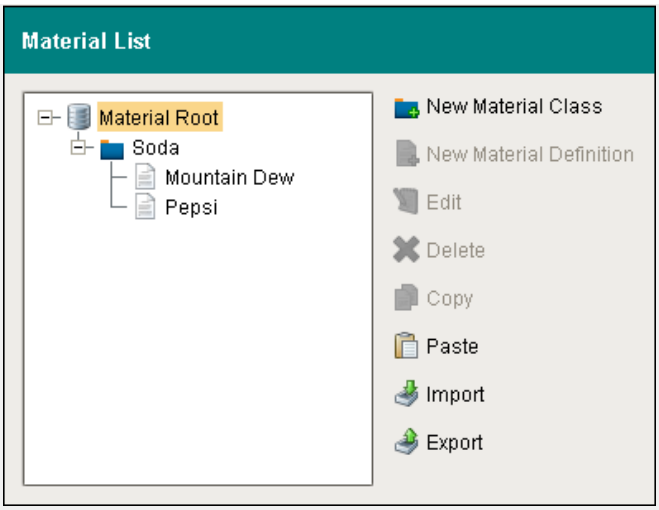
Custom Properties

The custom properties can be used to add user defined properties.



Example

The example below display a material class Soda with two material definitions: Mountain Dew and Pepsi.



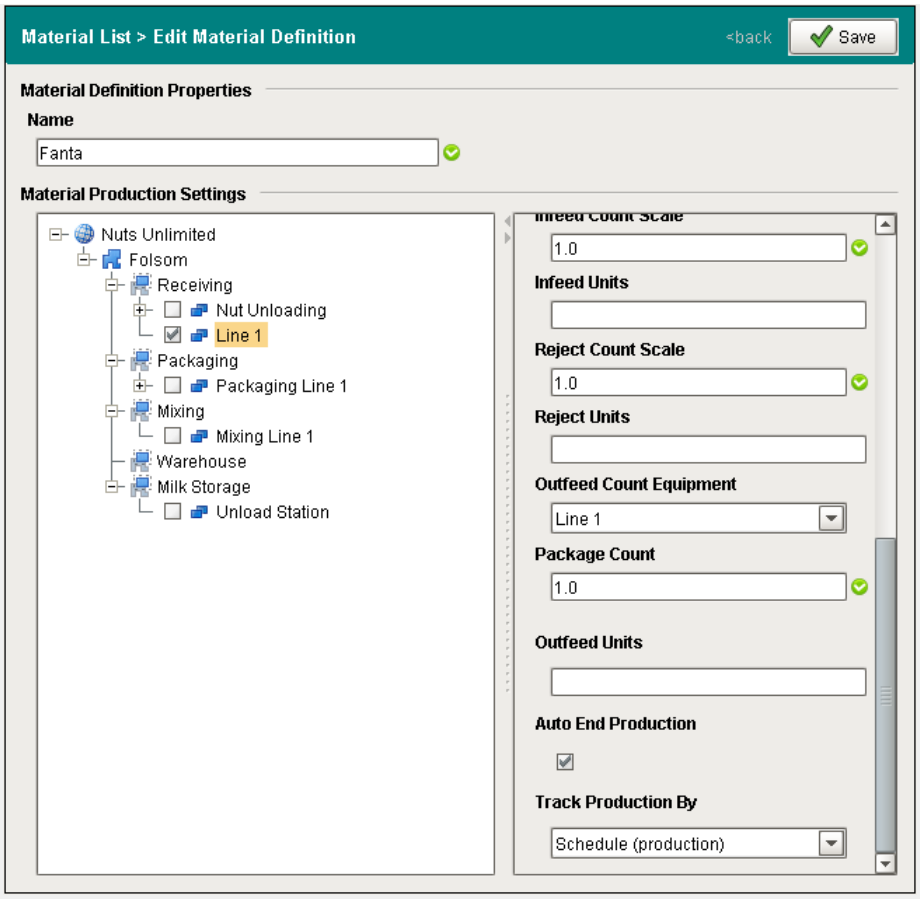
Property Name	Value
Title Background Color	0, 128, 128

If you click on Export, save window will appear to set your export location. Select the file type as xml to get the following xml file.

Material List Export

```
<?xml version="1.0" encoding="UTF-8"?>
-<MaterialRoot>

-<MaterialClass>
<Name>Soda</Name>
<Creator>OEE</Creator>

-<MaterialDef>
<Name>Mountain Dew </Name>
<Creator>OEE</Creator>
</MaterialDef>

-<MaterialDef>
<Name>Pepsi</Name>
<Creator>OEE</Creator>
</MaterialDef>
</MaterialClass>
</MaterialRoot>
```

To add a new material definition, first select the material class Soda and then **New Material Definition**. The following editor appears.



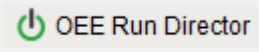
Give it a name. In this example, the N**ame** is set to Fanta, **Material Production Settings** to Line 1 and **Auto End Production** to True. Click Save.

Material classes can be added to an existing material class or to the material root.

OEE Run Director

General

Component Palette Icon:



In this Page

- Custom Properties

Description

OEE Run Director component is used to start and stop the production runs. Multiple operations can be run at a time by setting the Enable Simultaneous Active property to True. The  button will begin the production run,  button is for changeover and  button will end the run.

Properties

Name	Scripting	Category	Property Type	Description
Selection Mode	selectionMode	Data	int	The selection mode.
Equipment Path	equipmentPath	Data	String	The equipment path to s available segments. ⚠ If you copy and paste an equipme path to the Run Director compone Equipment Path property, the valu does not persist after saving, closing the windo and re-opening it This is by design. Otherwise everytime a wind

Name	Scripting	Category	Property Type	Description
				is opened, it will cause calls to the server which are not needed. Bindings, however, do persist. As an example, the property may be bound to a tag or a root container's custom property containing a path.
Enable Simultaneous Active	enableSimultaneousActive	Data	Boolean	If true, allows multiple operations to be active a the same time.
Previous Product Indexed State	previousProductIndexed	Hidden	Boolean	Whether or not the prev product has been index the next cell.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

loadIcon

- Description

Provides a chance to change an icon. Based on the icon name parameter, return the image path to the icon to use in place of the default icon.

- Parameters

self - A reference to the component that is invoking this function.

iconName - The name of the icon.

- Returns

Nothing

- Scope

Client

Example

```
# This example will return a path to a different image to
replace the default delate image:
if iconName == 'remove':/n/t/treturn 'Builtin/icons/24
/delete2.png'
```

Event Handlers

propertyChange

propertyChange

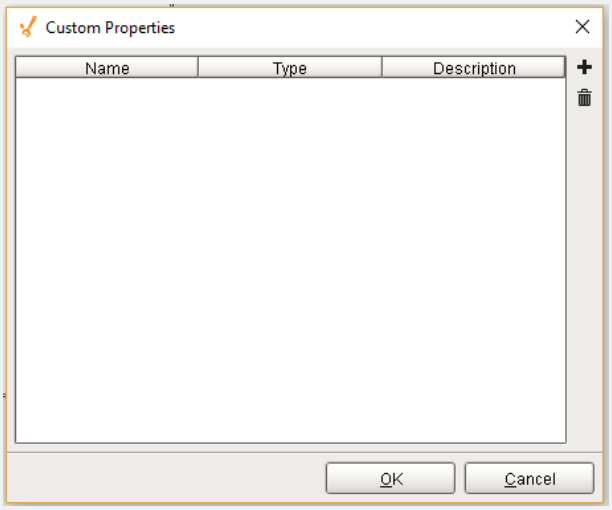
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

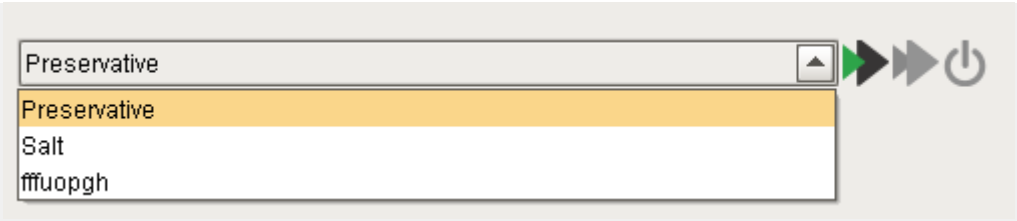
Customizers

Custom Properties

The custom properties can be used to add user defined properties.



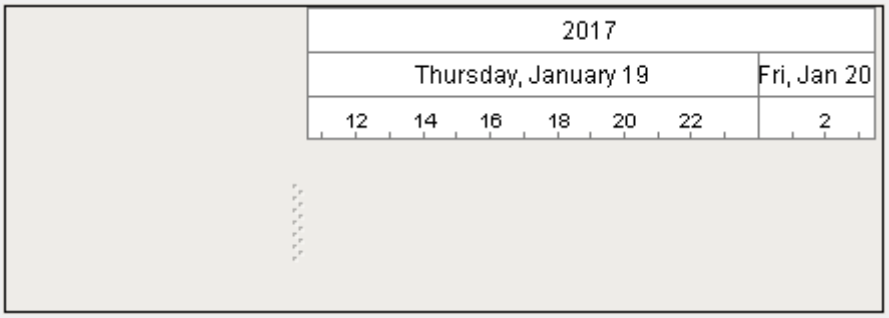
Examples



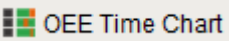
Property Name	Value
Equipment Path	[global]\Enterprise\Site 1\Area\Line 1
Selection Mode	Material

OEE Time Chart

General



Component Palette Icon:



In this Page

Description

A component that displays the line and cell downtime events of a run in a visual time chart.

Properties

Name	Scripting	Category	Property Type	Description
Line Path	linePath	Data	String	The production line path to show schedules.
Start Date	startDate	Data	Date	The start date to get schedules for the production line.
End Date	endDate	Data	Date	The end date to get schedules for the production line.
Update Interval	updateInterval	Data	int	The interval in seconds to update the time chart.
Show Ignition Schedule	showIgnitionSchedule	Data	Boolean	If true, show Ignition schedules.
Show Production Schedule	showProductionSchedule	Data	Boolean	If true, show production schedules.
Show Equipment Mode	showEquipmentMode	Data	Boolean	If true, show equipment modes.

Name	Scripting	Category	Property Type	Description
Show Equipment State	showEquipmentState	Data	Boolean	If true, show equipment states.
Show Equipment State	showEquipmentState	Data	Boolean	If true, show equipment states.
Ignition Schedule Color	ignitionScheduleColor	Appearance	Color	The foreground color of the ignition schedule.
Production Schedule Color	productionScheduleColor	Appearance	Color	The foreground color of the production schedule.
[Mode] Production Color	modeProductionColor	Appearance	Color	The foreground color of the Production mode.
[Mode] Changeover Color	modeChangeoverColor	Appearance	Color	The foreground color of the Changeover mode.
[Mode] Maintenance Color	modeMaintenanceColor	Appearance	Color	The foreground color of the Maintenance mode.

Name	Scripting	Category	Property Type	Description
[Mode] Other Color	modeOtherColor	Appearance	Color	The foreground color of the Other mode.
[Mode] Disabled Color	modeDisabledColor	Appearance	Color	The foreground color of the Disabled mode.
[State] Unplanned Downtime Color	stateUnplannedDowntimeColor	Appearance	Color	The foreground color of the Unplanned Downtime state.
[State] Planned Downtime Color	statePlannedDowntimeColor	Appearance	Color	The foreground color of the Planned Downtime state.
[State] Running Color	stateRunningColor	Appearance	Color	The foreground color of the Running state.
[State] Blocked Color	stateBlockedColor	Appearance	Color	The foreground color of the Blocked state.
	stateStarvedColor	Appearance	Color	

Name	Scripting	Category	Property Type	Description
[State] Starved Color				The foreground color of the Starved state.
[State] Idle Color	stateIdleColor	Appearance	Color	The foreground color of the Idle state.
[State] Disabled Color	stateDisabledColor	Appearance	Color	The foreground color of the Disabled state.
Row Height	rowHeight	Appearance	int	Row Height
[State] Unplanned Downtime Icon Path	unplannedDowntimeIconPath	Appearance	String	The relative path of an icon image to indicate the 'Unplanned Downtime' state. The recommended icon size is 16x16 pixels.
[State] Planned Downtime Icon Path	plannedDowntimeIconPath	Appearance	String	The relative path of an icon image to indicate the 'Planned Downtime'

Name	Scripting	Category	Property Type	Description
				state. The recommended icon size is 16x16 pixels.
[State] Running Icon Path	runningIconPath	Appearance	String	The relative path of an icon image to indicate the 'Running' state. The recommended icon size is 16x16 pixels.
[State] Blocked Icon Path	blockedIconPath	Appearance	String	The relative path of an icon image to indicate the 'Blocked' state. The recommended icon size is 16x16 pixels.
[State] Starved Icon Path	starvedIconPath	Appearance	String	The relative path of an icon image to indicate the 'Starved' state. The recommended icon size is 16x16 pixels.
[State] Idle Icon Path	idleIconPath	Appearance	String	

Name	Scripting	Category	Property Type	Description
				The relative path of an icon image to indicate the 'Idle' state. The recommended icon size is 16x16 pixels.
[State] Disabled Icon Path	disabledIconPath	Appearance	String	The relative path of an icon image to indicate the 'Disabled' state. The recommended icon size is 16x16 pixels.
Key Cell Icon Path	keyCellIconPath	Appearance	String	The relative path of an icon image to indicate the key cell. The recommended icon size is 16x16 pixels.
Collapse Icon Path	collapseIconPath	Appearance	String	The relative path of an icon image to indicate the collapse

Name	Scripting	Category	Property Type	Description
				status. The recommended icon size is 16x16 pixels.
Expand Icon Path	expandIconPath	Appearance	String	The relative path of an icon image to indicate the expand status. The recommended icon size is 16x16 pixels.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

getCustomField

- Description

Called to get a custom field.

- Parameters

self - A reference to the component that is invoking this function

equipmentPath - The equipment path as a string.

customField - The custom field object to set property values and return.

- Return

The custom field object.

- Scope

Client

Example

```
from java.awt import Color
    if equipmentPath == '[global]\\My Enterprise\\My
Site\\My Area\\My Line':
        customField.setMessage(prodCode)
        customField.setForeground(Color.RED)
        customField.setBackground(Color.YELLOW)
        customField.setIconPath('Sepasoft/Icons/product
code icon 16.png')
    return customField
    else:
        return None
```

loadIcon

- Description

Provides a chance to change an icon. Based on the icon name parameter, return the image path to the icon to use in place of the default icon.

- Parameters

self - A reference to the component that is invoking this function.

iconName - The name of the icon.

- Returns

Nothing

- Scope

Client

Example

```
# This example will return a path to a different image to
replace the default delete image:
if iconName == 'remove':/n/t/treturn 'Builtin/icons/24
/delete2.png'
```

Event Handlers

propertyChange

propertyChange

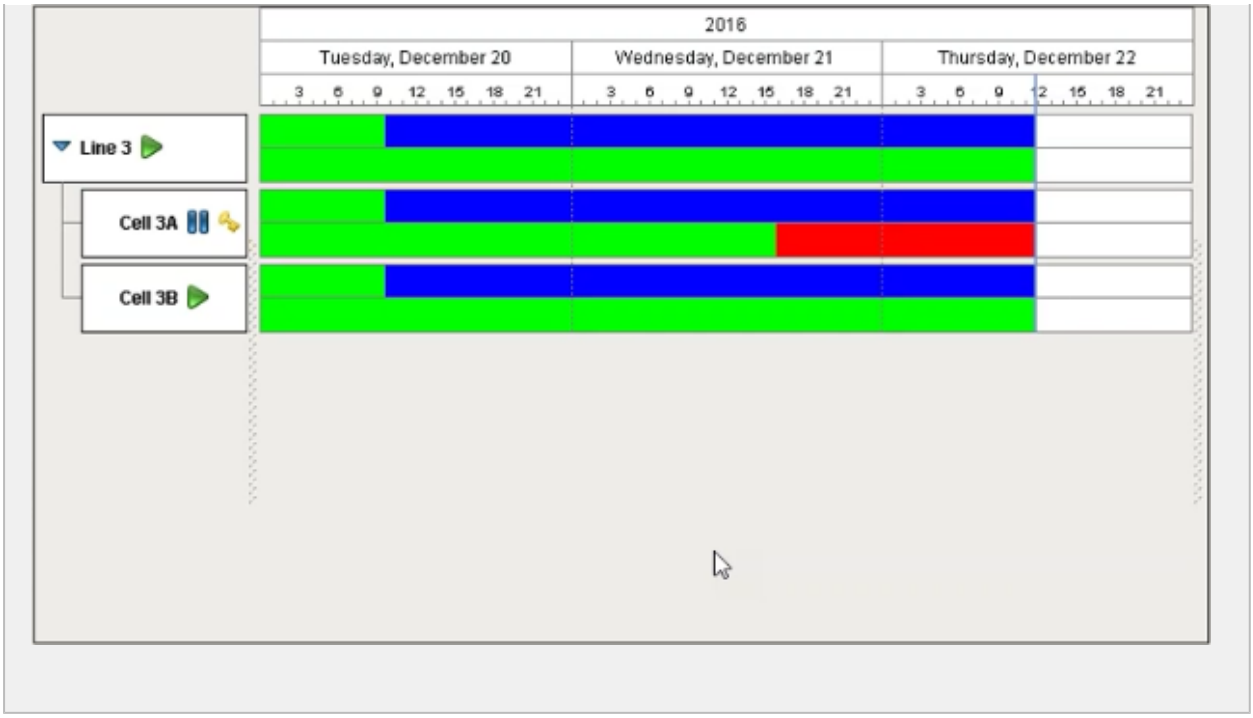
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

This component does not have any custom properties.

Examples

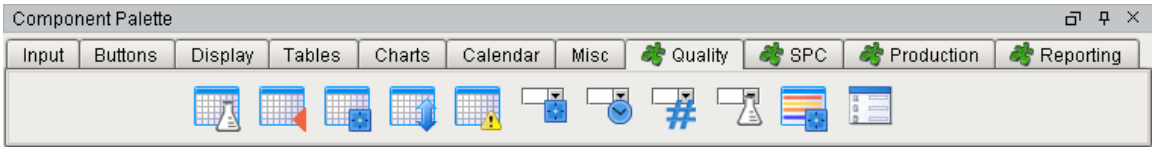


9.5.4 SPC Components

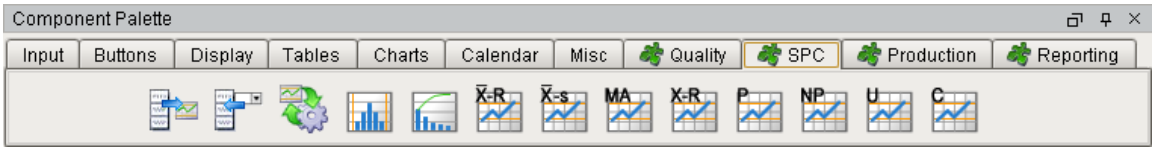
The SPC module provides a set of components to ensure quality, satisfy customer needs, and drive improvements.

There are two tabs in the designer that contain the SPC components.

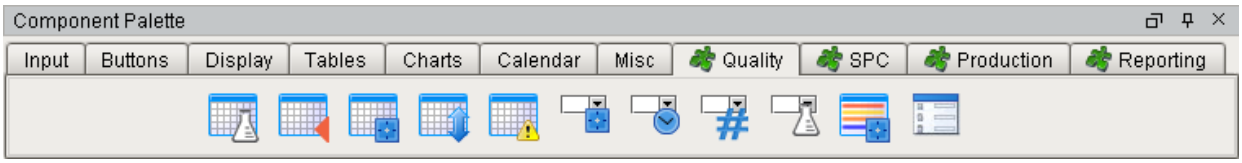
- The Quality tab contains the components used to create and capture sample data.



- The SPC tab contains the components used for sample analysis



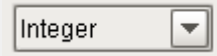
Quality Component Tab



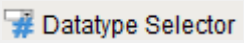
Quality Component

Datatype Selector

General



Component Palette Icon:



Description

A component that allows selection of sample attribute data types. The data types are built into the SPC module and cannot be added to or changed. There is no need for SQL queries or scripting to display the data types.

Properties

Name	Scripting	Category	Property Type	Description
Enabled	componentEnabled	Common	Boolean	If disabled, a component cannot be used.
Selected DataType	selectedDataType	Data	AttributeDataType	The Datatype of the currently selected type.
Styles	styles	Appearance	Dataset	Contains the component's styles.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

focus

focusGained

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it.

Property	Description
source	The component that fired this event.
oppositeComponent	The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vise versa.

focusLost

This event occurs when a component that had the input focus lost it to another component.

Property	Description
source	The component that fired this event.
oppositeComponent	

Property	Description
	The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vise versa.

key

keyPressed

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

keyReleased

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

keyTyped

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.

Property	Description
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

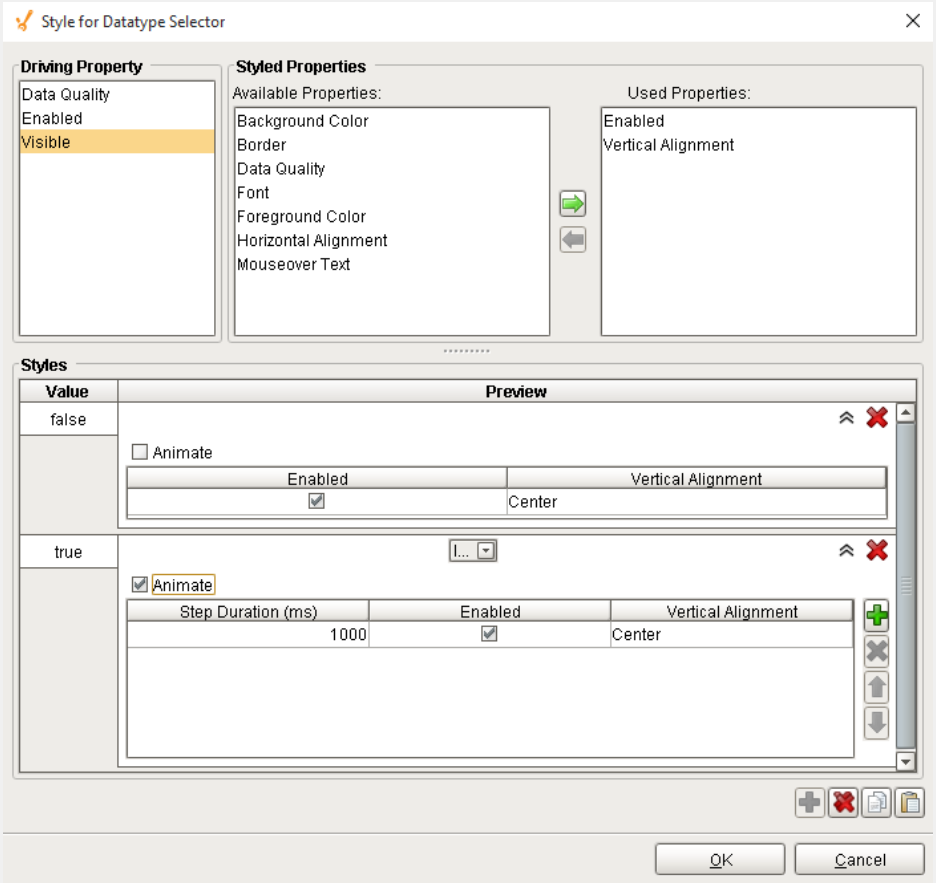
propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

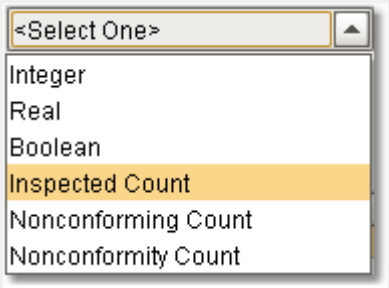
Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

This component does not have a customizer however this component relies on custom styles. The example below has the styles defined here:



Examples



SPCDataTypeSelector

Data Type Selector

Definition Attribute List

General

Name	Description	Data Type	Format	Max Value	Minimum Value	Default Value	Enabled	Required	Order

Component Palette Icon:  Definition Attribute List

Description

A component that provides a list of measurement attributes associated with a sample definition.

Properties

Name	Scripting	Category	Property Type	Description
Data	data	Data	DataSet	The data for the table.
Read Only	readOnly	Data	Boolean	If true, no editing is possible.
Show Disabled	showDisabled	Data	Boolean	If true, show disabled attributes.
Attribute Name	attName	Data	String	Name of the Attribute.
Odd Row Background	oddBackground	Appearance	Color	

Name	Scripting	Category	Property Type	Description
				The color which odd rows will be colored if background mode is 'Alternating'.
Table Foreground Color	tableFg	Appearance	Color	The color of the foreground for the table.
Table Background Color	tableBg	Appearance	Color	The color of the foreground for the table.
Row Height	rowHeight	Appearance	int	The height of each row, in pixels.
Selection Background	selectionBackground	Appearance	Color	The background color of a selected cell in the dropdown list.
Column Attributes Data	columnAttributesData	Data	Dataset	The dataset describing the column attributes.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

editSampleAttribute

add

Property	Description
source	The component that fired this event.
sampleAttrName	The currently selected sample definition attribute name.

edit

Property	Description
source	The component that fired this event.
sampleAttrName	The currently selected sample definition attribute name.

remove

Property	Description
source	The component that fired this event.
sampleAttrName	T he currently selected sample definition attribute name.

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	

Property	Description
	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

Property	Description
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	

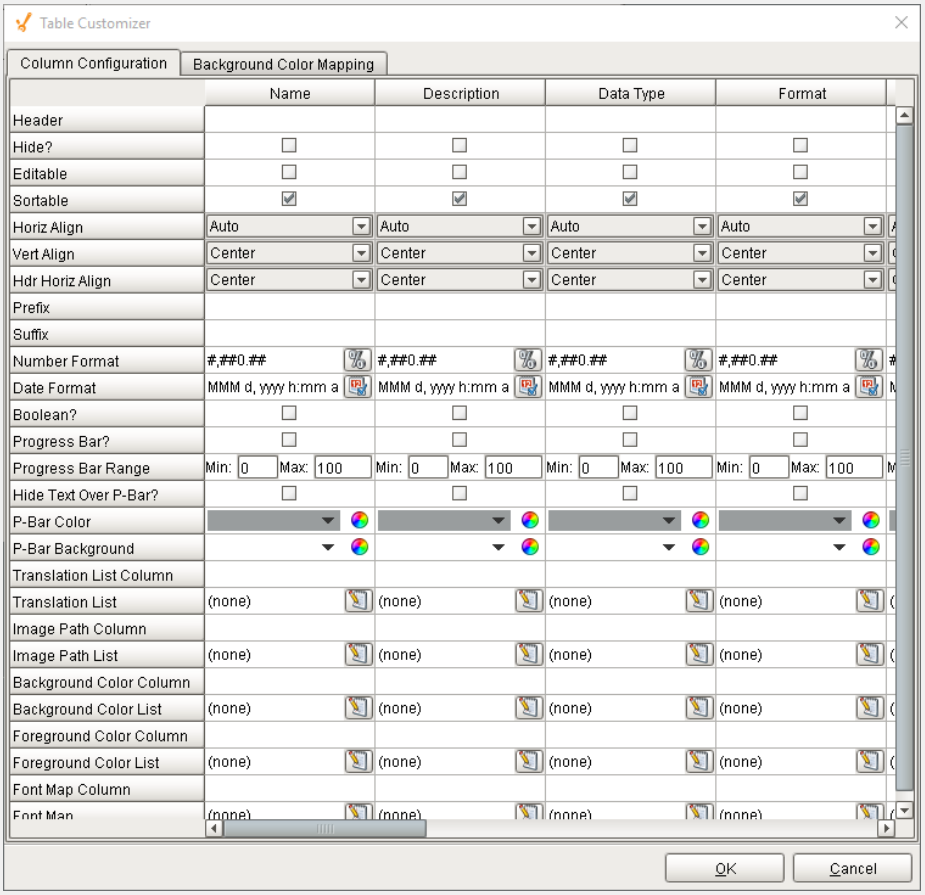
Property	Description
	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

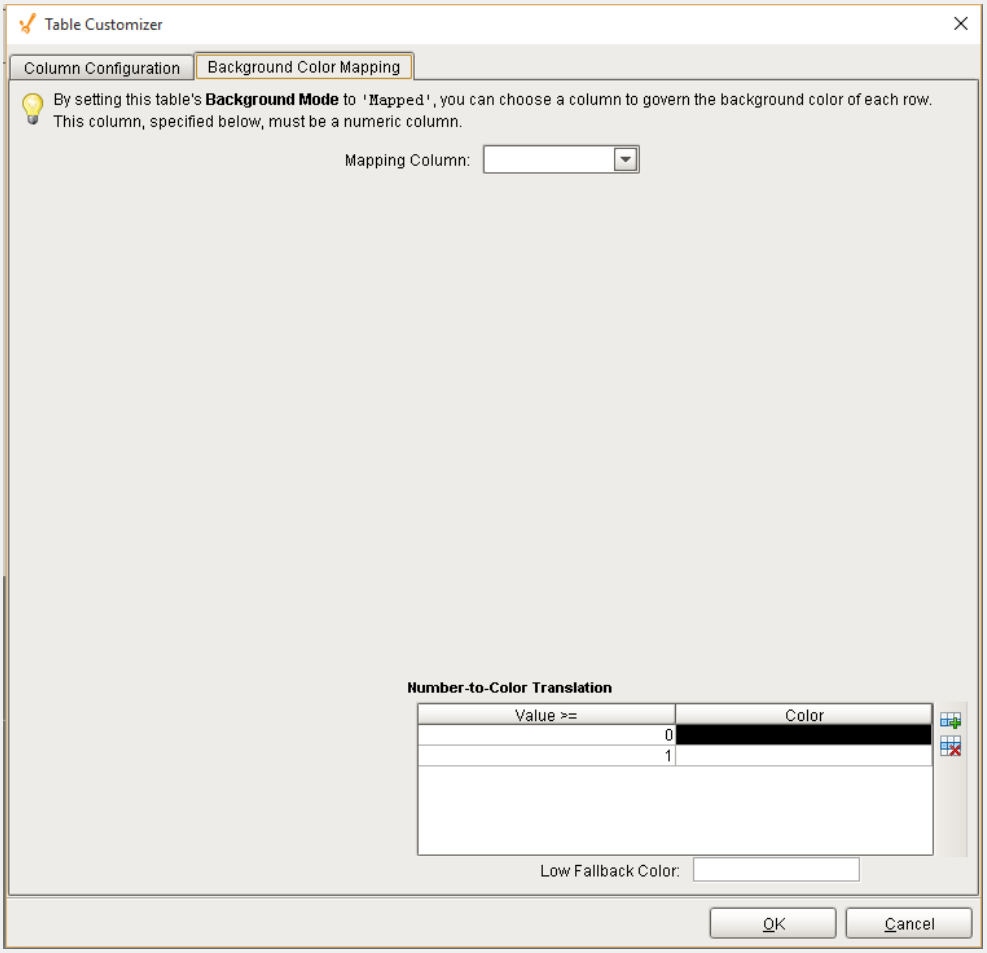
Table Customizer

Table Customizer manages the data entered into the Definition Attribute List.

Column Configuration



Background Color Mapping



Examples

There is no need for SQL queries or scripting to display sample definition attributes. If the Definition List component is on the same screen, the Definition Attribute List will find the Definition List component and register as a listener. Anytime the sample definition changes or the users selects a different sample definition, the Definition Attribute List the attributes will be updated automatically.

Name	Description	Data Type	Enabled	Required
Total Inspected		Inspected Count	☑	☑
Speck		Nonconforming Co...	☑	☑
Scratch		Nonconforming Co...	☑	☑
Hole		Nonconforming Co...	☑	☑
Discoloration		Nonconforming Co...	☑	☑

SPCDefAttributeList

Sample Definition Attribute List

The Ignition table customizer is used to change the appearance of the table. To access the customizer, right click on the Definition Attribute List component and select the Cutomizers-Table Customizer menu item. Using the customizer, you can hide columns, change colors, and change formatting to make the Definition Attribute List appear as desired.

When the Read Only property is set to false, the Move Up and Move Down menu items will appear in the popup menu. This allows the user to change the order that attributes appear in the Sample Entry component.

Definition Control Limit List

General

Name	Enable

Component Palette Icon:  Definition Control Limit List

Description

A component that provides a list of control limits to apply to a sample definition. All control limits that are configured in the project will appear in the list and can be selected by the user. Control limits that are selected by the user will be available to show on control charts and may be used during automatic signal evaluation.

Properties

Name	Scripting	Category	Property Type	Description
Row Height	rowHeight	Appearance	int	The height of each row, in pixels.
Odd Row Background	oddBackground	Appearance	Color	

Name	Scripting	Category	Property Type	Description
				The color which odd rows will be colored if background mode is 'Alternating'.
Selection Background	selectionBackground	Appearance	Color	The background color of a selected cell.
Selection Foreground	selectionForeground	Appearance	Color	The foreground color of a selected cell.
Show Horizontal Grid Lines?	showHorizontalLines	Appearance	boolean	Displays horizontal gridlines making it easier to read.
Show Vertical Grid Lines?	showVerticalLines	Appearance	boolean	Displays vertical gridlines making it easier to read.
Grid Line Color	gridColor	Appearance	Color	The color used to draw grid lines.
Selected Column	selectedColumn	Data	int	The index of the first selected column, or -1 if none.
Selected Row	selectedRow	Data	int	The index of the first selected row, or -1 if none.
Read Only	readOnly	Data	Boolean	Allow editing of table.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

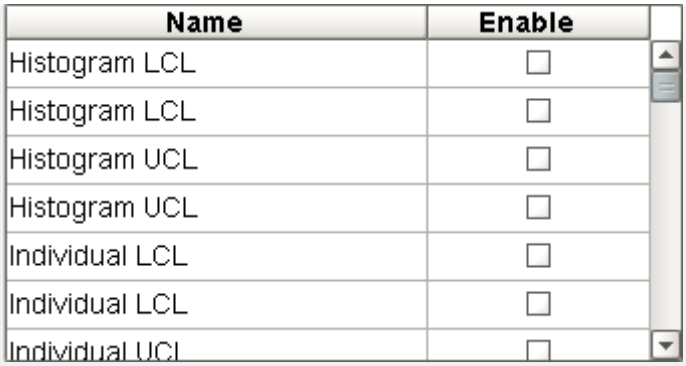
Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

This component does not have any custom properties.

Examples

There is no need for SQL queries or scripting to display control limits. If the Definition List component is on the same screen, the Definition Control Limit List will find the Definition List component and register as a listener. Anytime the sample definition changes or the users selects a different sample definition, the Definition Control Limit List will be updated automatically.



Name	Enable
Histogram LCL	☐
Histogram LCL	☐
Histogram UCL	☐
Histogram UCL	☐
Individual LCL	☐
Individual LCL	☐
Individual UCL	☐

SPCDefinitionCLList

Sample Definition Control Limit List

Definition List

General

Name	Description	Version	DefUUID

Component Palette Icon: Definition List

Description

A component that provides a list of sample definitions. A sample definition defines the attributes (measurements), locations, control limits and out of control signals to use for samples. It allows for adding, editing and deleting samples and works with the Definition Attribute List, Definition Location List, Definition Control Limit List and Definition Signals List components.

Properties

Name	Scripting	Category	Property Type	Description
Activity Timeout	activityTimeout	Data	Int4	Number of seconds to wait after user activity before update.
Data	data	Data	DataSet	The data for the table.
Read Only	readOnly	Data	Boolean	No editing is possible.
Show Disabled	showDisabled	Data	Boolean	Shows the disabled test definitions.
Odd Row Background	oddBackground	Appearance	Color	The color which odd rows will be colored if background mode is 'Alternating'.
Table Foreground Color	tableFg	Appearance	Color	The color of the foreground for the table.
Table Background Color	tableBg	Appearance	Color	The color of the foreground for the table.
Row Height	rowHeight	Appearance	int	The height of each row, in pixels.
Selection Background	selectionBackground	Appearance	Color	The background color of a selected cell in the dropdown list.
Column Attributes Data	columnAttributesData	Data	Dataset	The dataset describing the column attributes.

Scripting

Scripting Functions

save

- Description

Save changes to the currently selected sample definition .

- Parameters

None

- Return

Nothing

- Scope

Client

cancel

- Description

Undo the changes to the currently selected sample definition.

- Parameters

None

- Return

Nothing

- Scope

Client

getSampleDefinition

- Description

Return the currently selected sample definition.

- Parameters

None

- Return

The currently selected sample definition.

- Scope

Client

addSampleDefinition

- Description

Add the sample definition specified in the parameter.

- Parameters

None

- Return

Nothing

- Scope

Client

updateSampleDefinition

- Description

Update the sample definition specified in the parameter.

- Parameters

None

- Return

Nothing

- Scope

Client

refresh

- Description

Refresh the currently selected sample definition. This causes any associated components such as the Definition Attribute List to also be refreshed.

- Parameters

None

- Return

Nothing

- Scope

Client

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

editSampleDefinition

This event fires when the mouse enters the space over the source component.

add

Property	Description
source	The component that fired this event.
sampleDefinitionName	Name of the sample definition of this event.

edit

Property	Description
source	The component that fired this event.
sampleDefinitionName	Name of the sample definition of this event.

remove

Property	Description
source	The component that fired this event.
removeDefinition	Removes the sample definition.

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	

Property	Description
	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

Property	Description
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	

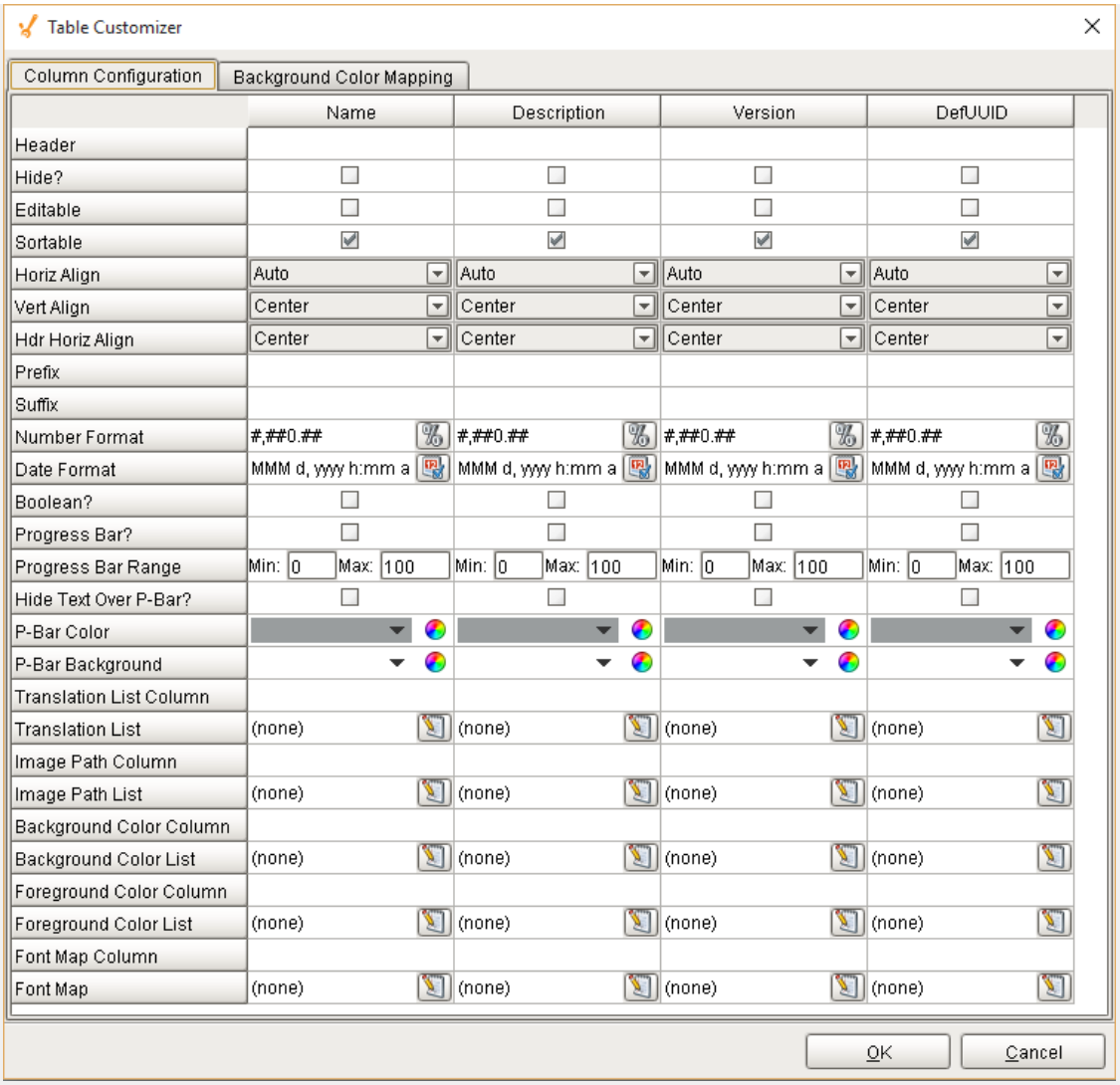
Property	Description
	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

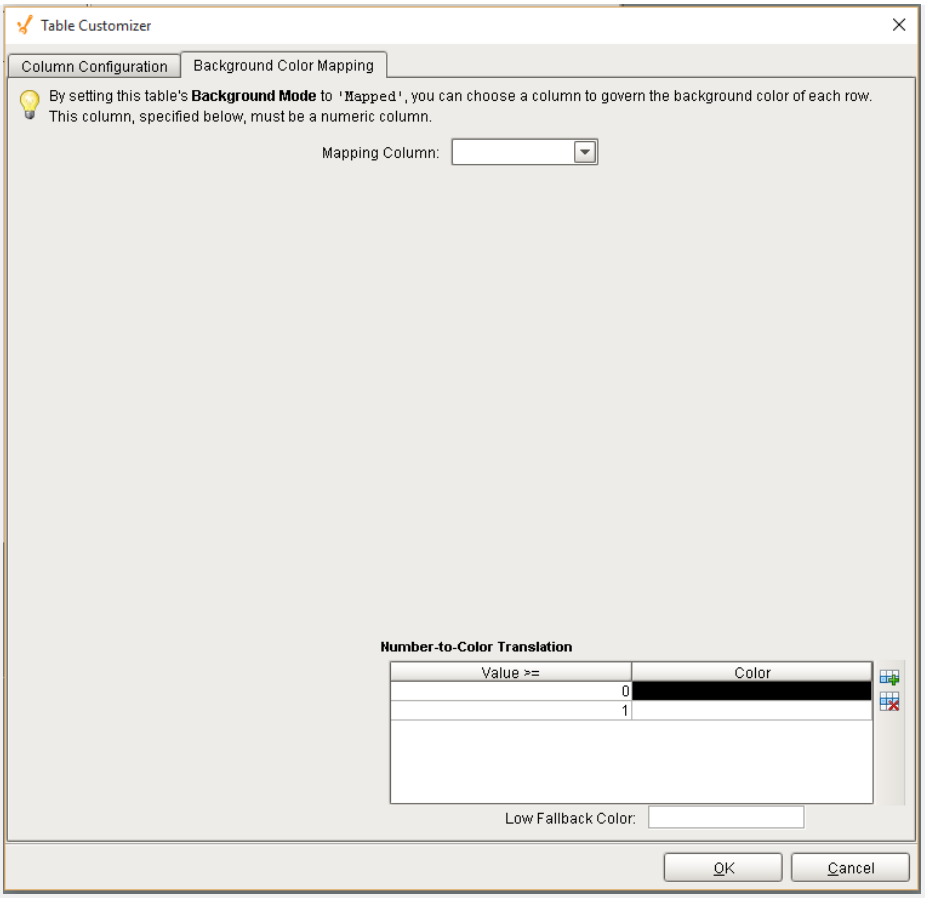
Table Customizer

Table Customizer manages the data entered into the Definition List component.

Column Configuration



Background Color Mapping



Examples

There is no need for SQL queries or scripting to display sample definitions. The SPC Module will send notifications to each client with a Definition List component being displayed when there is a change to any sample definitions made by another user. This event-based functionality optimizes updates, reducing database updates and network bandwidth.

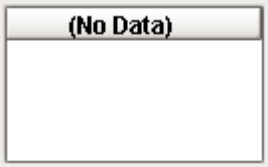
Name	Version
Final	1
Final Imperfections	1
pH	1
pH2	1
Product Moisture	1
Value Inspection	1
Viscosity	1

Sample Definition List

The Ignition table customizer is used to change the appearance of the table. To access the customizer, right-click on the Definition List component and select the Cutomizers->Table Customizer menu item. Using the customizer, you can hide columns, change colors, and change formatting to make the Definition List appear as desired.

Definition Location List

General



Component Palette Icon: Definition Location List

Description

A component that provides a list of production locations that a sample can be taken from for the associated sample definition. In other words, a test is defined (sample definition) and it has locations that are appropriate to take the test at (production location). There is no need for SQL queries or scripting to display allowable locations. If the Definition List component is on the same screen, the Definition Location List will find the Definition List component and register as a listener. Anytime the sample definition changes or the users selects a different sample definition, the Definition Location List will be updated automatically.

Properties

Name	Scripting	Category	Property Type	Description
Data	data	Data	DataSet	The data for the table.
Read Only	readOnly	Data	Boolean	No editing is possible.

Name	Scripting	Category	Property Type	Description
Show Disabled	showDisabled	Data	Boolean	Show disabled locations.
Include Removed	includeRemoved	Data	Boolean	Includes the removed locations.
Location ID	locationID	Data	int	The ID for the location.
Odd Row Background	oddBackground	Appearance	Color	The color which odd rows will be colored if background mode is 'Alternating'.
Table Foreground Color	tableFg	Appearance	Color	The color of the foreground for the table.
Table Background Color	tableBg	Appearance	Color	The color of the foreground for the table.
Row Height	rowHeight	Appearance	int	The height of each row, in pixels.
Column Attributes Data	columnAttributesData	Data	Dataset	The dataset describing the column attributes.
Selection Background	selectionBackground	Appearance	Color	The background color of a selected cell in the dropdown list.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

editSampleLocation

add

Property	Description
source	The component that fired this event.
sampleLocName	Name of the sample definition location.

edit

Property	Description
source	The component that fired this event.
sampleLocName	Name of the sample definition location.

remove

Property	Description
source	The component that fired this event.
sampleLocName	Name of the sample definition location.

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

Property	Description
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

Property	Description
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	

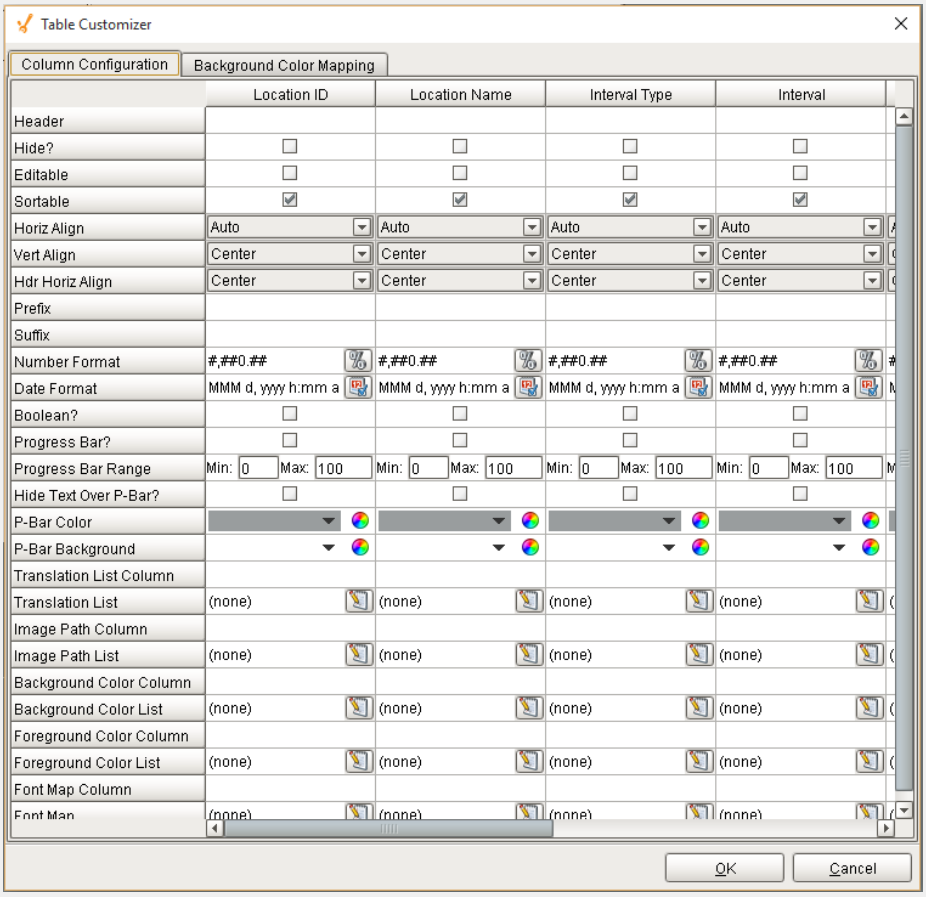
Property	Description
	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

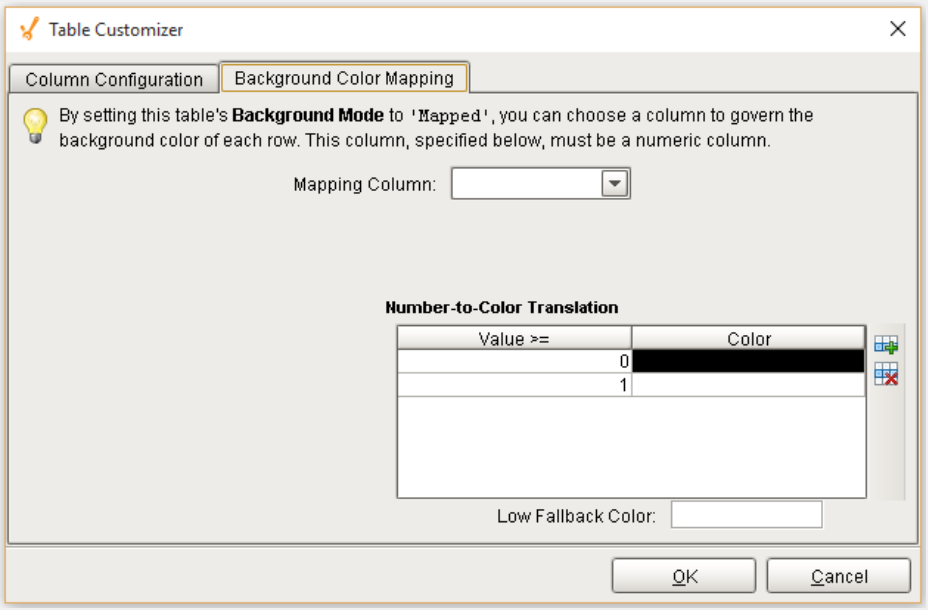
Table Customizer

Table Customizer manages the data entered into the Definition Location List.

Column Configuration



Background Color Mapping



Examples

Location Name	Interval Type	Interval	Auto Approve	Enabled
Line 1 Quality	Manual	0	☑	☑
Line 2 Quality	Manual	0	☑	☑

SPCDefinitionLocationList

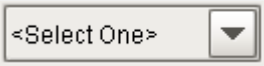
Sample Definition Location List

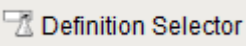
The Ignition table customizer is used to change the appearance of the table. To access the customizer, right-click on the Definition Location List component and select the Cutomizers->Table Customizer menu item. Using the customizer, you can hide columns, change colors, and change formatting to make the Definition Location List appear as desired.

When the Read Only property is set to false, the Move Up and Move Down menu items will appear in the popup menu. This allows the user to change the order that attributes appear in the Sample Entry component.

Definition Selector

General



Component Palette Icon: 

Description

A component that allows selection of sample definitions. One source of sample definitions is from the definition management screen that uses the Definition List component. There is no need for SQL queries or scripting to display the data types.

Properties

Name	Scripting	Category	Property Type	Description
Enabled	componentEnabled	Common	Boolean	If disabled, a component cannot be used.
Location Path	locationPath	Data	String	The path of the production location item.
Tag	tag	Data	String	The tag to limit the sample definitions by.

Selected Sample Definition Name	selectedSampleDefinitionName	Data	String	The name of the currently selected sample definition type.
Selected Sample DefUUID	selectedSampleDefUUID	Data	String	The defUUID of the currently selected sample definition type.
Styles	styles	Appearance	Dataset	Contains the component's styles.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

focus

focusGained

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it.

Property	Description
source	The component that fired this event.
oppositeComponent	The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vise versa.

focusLost

This event occurs when a component that had the input focus lost it to another component.

Property	Description
source	The component that fired this event.
oppositeComponent	The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vise versa.

key

keyPressed

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.

Property	Description
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

keyReleased

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.

Property	Description
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

keyTyped

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	

Property	Description
	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

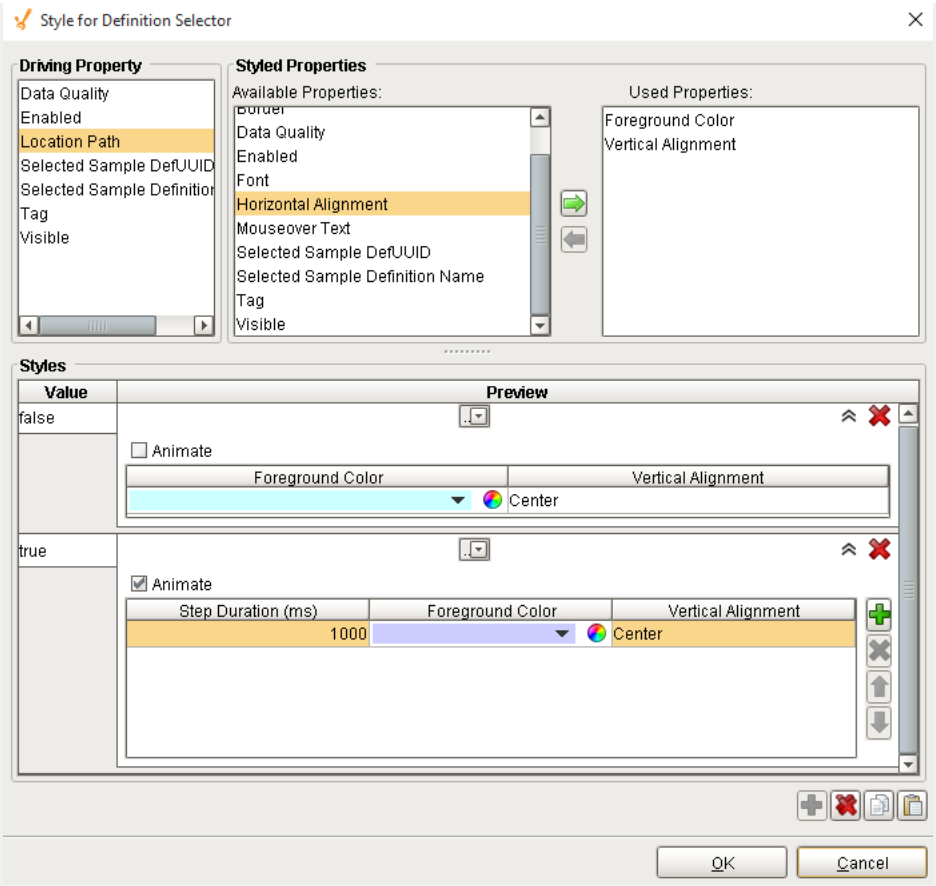
propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

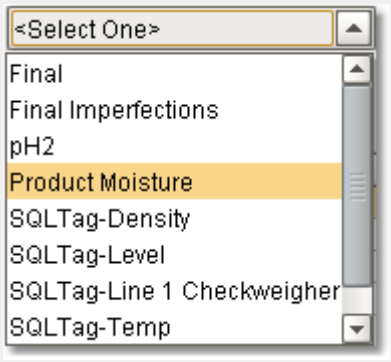
Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

This component does not have a customizer however this component relies on custom styles. The example below has the styles defined here:



Examples



SPCDefinitionSelector

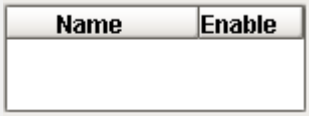
Sample Definition Selector

When an allowable location is added to a sample definition, a tag value can be set. This component can limit the sample definitions that appear by entering in matching tag values. It is typically used for defining who has ownership for collecting sample data. For example, the lab takes samples at packaging line 1 every 2 hours. The operator also takes samples

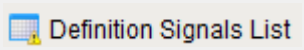
at packaging line 1 every 1 hour. When the lab takes a sample, they don't want to see information that the operator has ownership for and visa versa. To accomplish this, set the tag value to "Lab" for sample definitions that the lab has ownership for and to "Operator" for sample definitions that the operator has ownership for.

Definition Signals List

General



Component Palette Icon:



Description

A component that provides a list of signals (rules) to apply to a sample definition. All signals that are configured in the project will appear in the list and can be selected by the user. Signals that are selected by the user will be available to show on control charts and will be automatically evaluated when new samples are added.

Properties

Name	Scripting	Category	Property Type	Description
Row Height	rowHeight	Appearance	int	The height of each row, in pixels.
Odd Row Background	oddBackground	Appearance	Color	The color which odd rows will be colored if background mode is 'Alternating'.

Name	Scripting	Category	Property Type	Description
Selection Background	selectionBackground	Appearance	Color	The background color of a selected cell.
Selection Foreground	selectionForeground	Appearance	Color	The foreground color of a selected cell.
Show Horizontal Grid Lines?	showHorizontalLines	Appearance	boolean	Displays horizontal gridlines making it easier to read.
Show Vertical Grid Lines?	showVerticalLines	Appearance	boolean	Displays vertical gridlines making it easier to read.
Grid Line Color	gridColor	Appearance	Color	The color used to draw grid lines.
Selected Column	selectedColumn	Data	int	The index of the first selected column, or -1 if none.
Selected Row	selectedRow	Data	int	The index of the first selected row, or -1 if none.
Read Only	readOnly	Data	Boolean	Allow editing of table.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	

Property	Description
	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description	
source	The component that fired this event.	
button	The code for the button that caused this event to fire.	
clickCount	The number of mouse clicks associated with this event.	
x	The x-coordinate (with respect to the source component) of this mouse event.	
y	The y-coordinate (with respect to the source component) of this mouse event.	
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.	
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.	
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.	
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.	

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

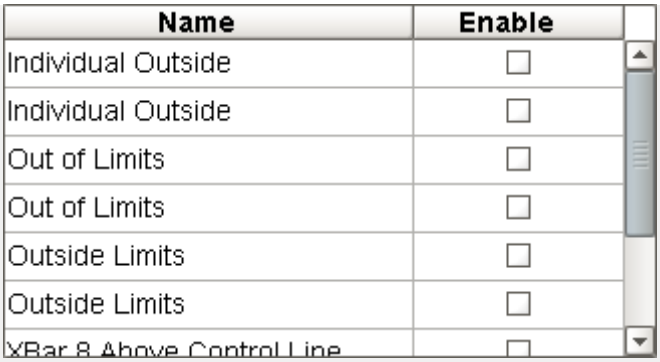
Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

This component does not have any custom properties.

Examples

There is no need for SQL queries or scripting to display signals. If the Definition List component is on the same screen, the Definition Signals List will find the Definition List component and register as a listener. Anytime the sample definition changes or the users selects a different sample definition, the Definition Signals List will be updated automatically.



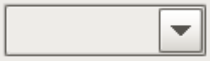
Name	Enable
Individual Outside	☐
Individual Outside	☐
Out of Limits	☐
Out of Limits	☐
Outside Limits	☐
Outside Limits	☐
XBar 8 Above Control Line	☐

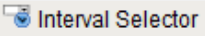
SPCDefinitionSignalList

Sample Definition Signal List

Interval Selector

General



Component Palette Icon: 

Description

A component that allows selection of sample intervals. All intervals that are configured in the project will appear in the list and can be selected by the user. See the Sample Intervals section for more information on intervals. There is no need for SQL queries or scripting to display intervals.

Properties

Name	Scripting	Category	Property Type	Description
Enabled	componentEnabled	Common	Boolean	If disabled, a component cannot be used.

Name	Scripting	Category	Property Type	Description
Selected Interval	selectedInterval	Data	String	The Interval of the currently selected interval type.
Styles	styles	Appearance	Dataset	Contains the component's styles.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

focus

focusGained

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it.

Property	Description
source	The component that fired this event.
oppositeComponent	

Property	Description
	The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vise versa.

focusLost

This event occurs when a component that had the input focus lost it to another component.

Property	Description
source	The component that fired this event.
oppositeComponent	The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vise versa.

key

keyPressed

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.

Property	Description
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

keyReleased

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

keyTyped

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.

Property	Description
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.

Property	Description
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.

Property	Description
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	

Property	Description
	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

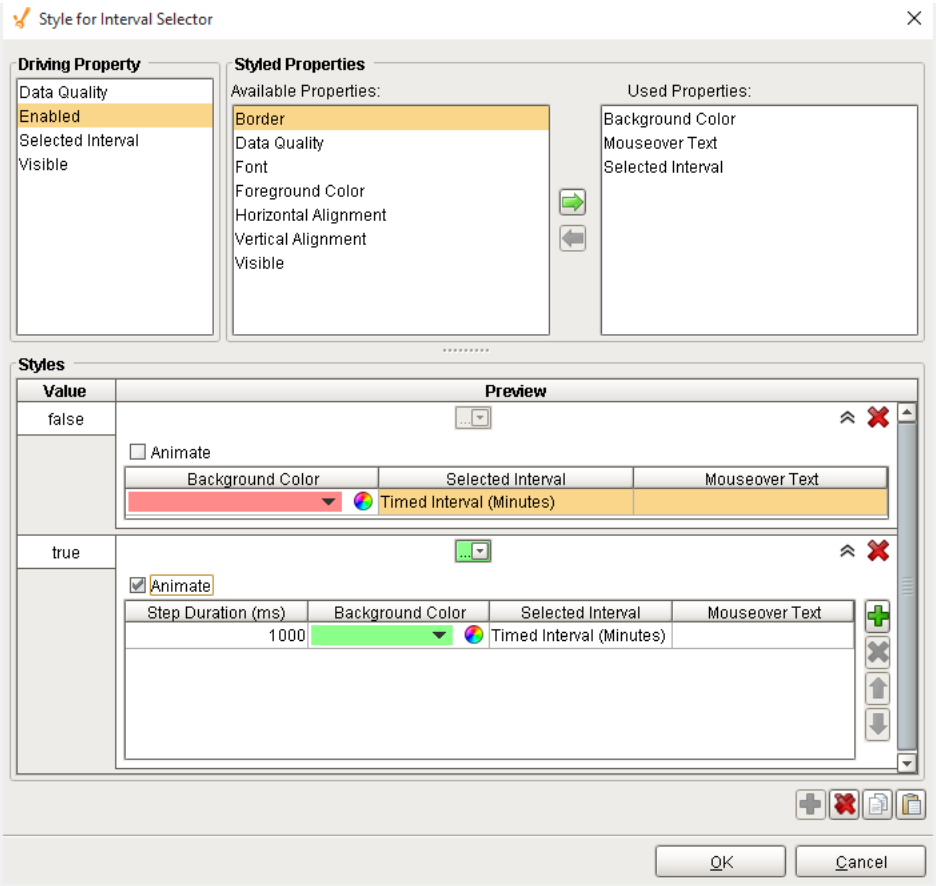
propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

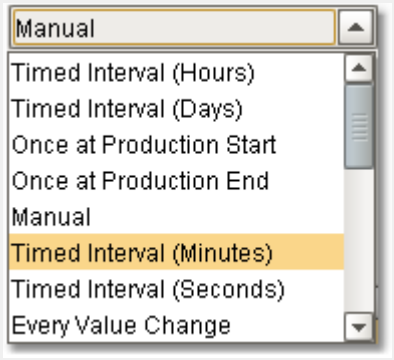
Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

This component does not have a customizer however this component relies on custom styles. The example below has the styles defined here:



Examples

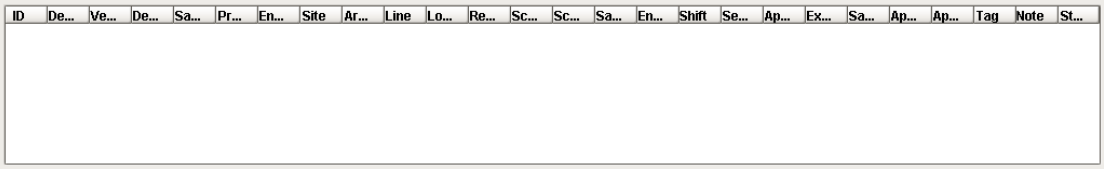


SPCIntervallSelector

Interval Selector

Location Sample List

General



Component Palette Icon: Location Sample List

Description

A component that displays samples for a location and optionally by sample ownership. Through configuration properties, it can show samples that are scheduled to be coming due, due, overdue, or waiting approval or approved. There is no need for SQL queries or scripting to display the samples.

Properties

Name	Scripting	Category	Property Type	Description
Header Font	headerFont	Appearance	Font	Font of the table's header text.
Header Foreground Color	headerForeground	Appearance	Color	The foreground color of the table's header.
Row Selection Allowed	rowSelectionAllowed	Behavior	boolean	This flag is used in conjunction with the Column Selection Allowed

Name	Scripting	Category	Property Type	Description
				flag to determine whether not whole-rows, whole-columns, or both.
Header Visible	headerVisible	Appearance	boolean	Whether or not the table header is visible.
Resizing Allowed	resizingAllowed	Behavior	boolean	Whether or not the user is allowed to resize table headers or not.
Row Height	rowHeight	Appearance	int	The height of each row, in pixels.
Odd Row Background	oddBackground	Appearance	Color	The color which odd rows will be colored if background mode is 'Alternating'.
Selection Background	selectionBackground	Appearance	Color	The background color of a selected cell.
Selection Foreground	selectionForeground	Appearance	Color	The foreground color of a selected cell.
Show Horizontal Grid Lines?	showHorizontalLines	Appearance	boolean	Displays horizontal gridlines making it easier to read.
	showVerticalLines	Appearance	boolean	

Name	Scripting	Category	Property Type	Description
Show Vertical Grid Lines?				Displays vertical gridlines making it easier to read.
Grid Line Color	gridColor	Appearance	Color	The color used to draw grid lines.
Column Attributes Data	columnAttributesData	Data	DataSet	The dataset describing the column attributes.
Selected Row	selectedRow	Data	int	The index of the first selected row, or -1 if none.
Data	data	Data	DataSet	Raw sample data.
Read Only	readOnly	Data	Boolean	No editing is possible.
Show Waiting Approval	showWaitingApproval	Data	Boolean	Show samples waiting for approval.
Show Approved Samples	showApprovedSamples	Data	Boolean	Show samples that have been approved.
Show Due Samples	showDueSamples	Data	Boolean	Show samples that are due.
Show Coming Due Samples	showComingDueSamples	Data	Boolean	Show samples that are coming due.

Name	Scripting	Category	Property Type	Description
Show Overdue Samples	showOverdueSamples	Data	Boolean	Show samples that are overdue.
Show Removed Samples	showRemovedSamples	Data	Boolean	Show samples that are flagged for removal.
Enable Note Editing	enableNoteEditing	Data	Boolean	Set to true, to allow users to add or edit sample notes.
Start Date	startDate	Data	DateTime	Start date of range to show sample for.
End Date	endDate	Data	DateTime	End date of range to show sample for.
Location Path	locationPath	Data	String	The path of the location to show samples for.
Product Code	productCode	Data	String	The product code to show samples for.
Reference No	referenceNo	Data	String	The reference number to show samples for.
Tag	tag	Data	String	The tag to show samples for.

Scripting

Scripting Functions

createByDefUUID

- Description

Create a new sample based on the sample definition specified by the defUUID parameter .

- Parameters

String defUUID - Sample definition UUID to base the new sample on. A UUID is a universally unique identifier that, once assigned to a sample definition, will never change. It is automatically generated when a sample definition is created and is unique in that no two samples definitions will have the same UUID.

- Return

Sample Object - An instance of a new sample.

- Scope

Client

createByDefName

- Description

Create a new sample based on the sample definition specified by the defName parameter .

- Parameters

String defName - Sample definition name to base the new sample on.

- Return

Sample Object - An instance of a new sample.

- Scope

Client

update

- Description

Create a new sample based on the sample definition specified by the defName parameter .

- Parameters

This is the sample to either update, if it already exists, or add, if it does not already exist.

- Return

String Message of any errors that may have occurred during the update operation.

- Scope

Client

exclude

- Description

Excludes the sample specified by uuid parameter.

- Parameters

String sampleUUID - The UUID to an existing sample to exclude.

- Return

String Message of any errors that may have occurred during the operation.

- Scope

Client

include

- Description

Includes the sample specified by uuid parameter.

- Parameters

String sampleUUID - The UUID to an existing sample to include.

- Return

String Message of any errors that may have occurred during the operation.

- Scope

Client

approve

- Description

Approve the sample specified by the sample parameter .

- Parameters

Sample Object - This is the sample to approve.

- Return

String Message of any errors that may have occurred during the approve operation.

- Scope

Client

unapprove

- Description

Unapprove the sample specified by the sample parameter .

- Parameters

Sample Object - This is the sample to unapprove.

- Return

String Message of any errors that may have occurred during the unapprove operation.

- Scope

Client

showEditNotePopup

- Description

Show the note popup to allow the user to add or edit the note tied to the currently selected sample.

- Parameters

None

- Return

Nothing

- Scope

Client

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

editSampleLocation

add

Property	Description
source	The component that fired this event.
sampleUUID	UUID of the sample.

edit

Property	Description
source	The component that fired this event.
sampleUUID	UUID of the sample.

remove

Property	Description
source	The component that fired this event.
sampleUUID	UUID of the sample.

approve

Property	Description
source	The component that fired this event.
sampleUUID	UUID of the sample.

unapprove

Property	Description
source	The component that fired this event.
sampleUUID	UUID of the sample.

review

Property	Description
source	The component that fired this event.
sampleUUID	UUID of the sample.

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	

Property	Description
	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

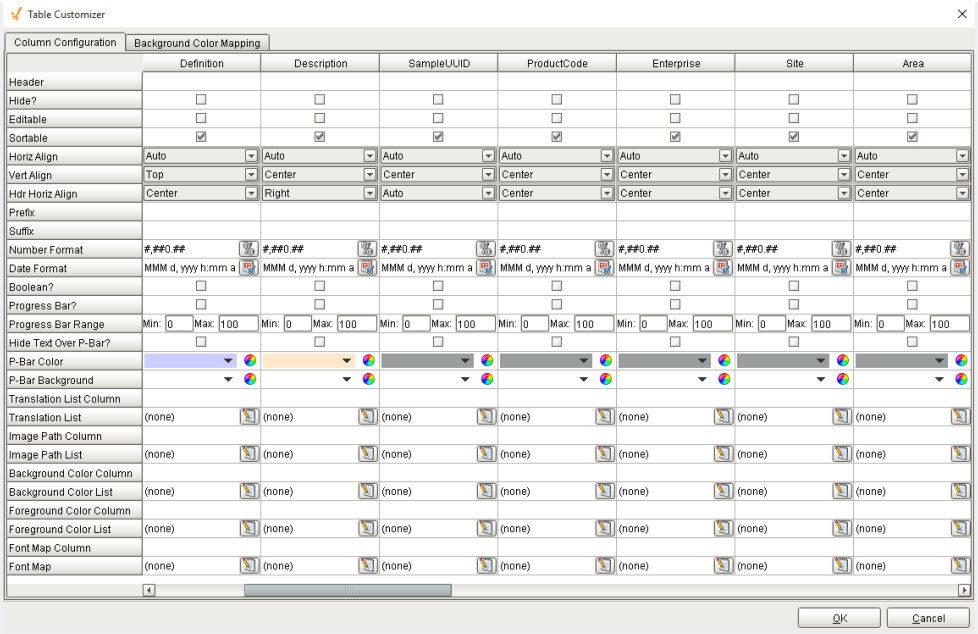
Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

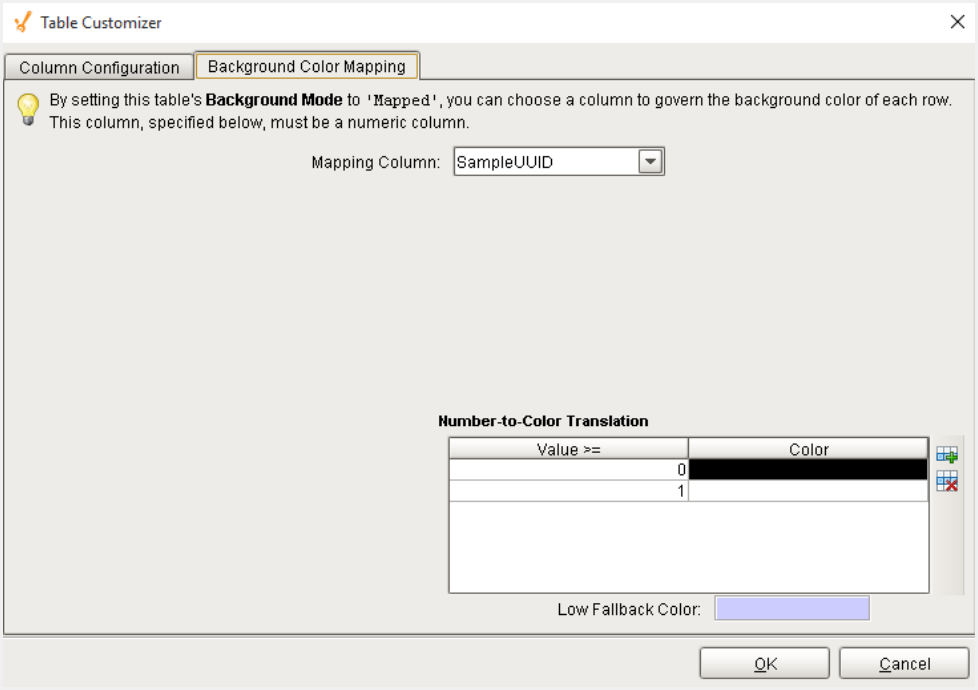
Table Customizer

Table Customizer manages the data entered into the Location Sample List component.

Column Configuration



Background Color Mapping



Next »

Examples

Sample Type	Product Code	Reference No	Scheduled Start	Taken Date Time
Value Inspection			Apr 26, 2012 9:14 AM	
Value Inspection			Apr 26, 2012 10:14 AM	
Value Inspection			Apr 23, 2012 9:11 AM	Apr 25, 2012 7:57 AM

Sample Definition Selector

When an allowable location is added to a sample definition, a tag value can be set. This component can limit the samples that appear by entering in matching tag values. It is typically used for defining who has ownership for collecting sample data. For example, the lab takes samples at packaging line 1 every 2 hours. The operator also takes samples at packaging line 1 every 1 hour. The lab does not want to see samples that the operator has ownership for and vice versa. To accomplish this, set the tag value to "Lab" for sample definitions that the lab has ownership for and to "Operator" for sample definitions that the operator has ownership for.

The Ignition table customizer is used to change the appearance of the table. To access the customizer, right-click on the Location Sample List component and select the Cutomizers->Table Customizer menu item. Using the customizer, you can hide columns, change colors, change formatting to make the Location Sample List appear as desired.

Location Selector

General

<Select One>

Component Palette Icon: Location Selector

Description

A component that allows selection of production locations. Production locations are defined in the production model using the Ignition Designer. See Production Model Overview for more information. There is no need for SQL queries or scripting to display locations. The selected location is reflected in Selected Location Name, Path and Location ID properties.

Properties

Name	Scripting	Category	Property	Description

Selection Mode	selectionMode	Behavior	int	The selection mode determines the behavior of the dropdown: whether its selected value must strictly be in the underlying set of choices, whether it is flexible, or even user-editable.
Location Name Filter	locationNameFilter	Data	String	Comma separated list of location names to display. Leave blank for all lines.
Production Model Item Path Filter	itemPathFilter	Data	String	The top level Production Model Item path to filter the lines to display. Leave blank to include all Production Model items.
Selected Location Name	selectedLocationName	Data	String	The name of the selected location.
Selected Location Path	selectedLocationPath	Data	String	The path of the currently selected location.
	selectedPathWithoutProject	Data	String	

Name	Scripting	Category	Property Type	Description
Selected Path Without Project				The path of the currently selected location without the leading project name.
Selected Location ID	selectedLocationID	Data	int	The ID of the currently selected location.
Display Path	displayPath	Appearance	Boolean	Displays the full path of the location.
Selection Background	selectionBackground	Appearance	Color	The background color of a selected cell in the dropdown list.
Max Table Width	maxTableWidth	Appearance	int	The maximum width allowed for the dropdown table.
Max Table Height	maxTableHeight	Appearance	int	The maximum height allowed for the dropdown table.
Styles	styles	Appearance	Dataset	Contains the component's styles.
Horizontal Alignment	horizontalAlignment	Layout	int	Determines the alignment of the contents along the X axis

Name	Scripting	Category	Property Type	Description
Vertical Alignment	verticalAlignment	Layout	int	Determines the alignment of the contents along the Y axis

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

focus

focusGained

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it.

Name	Scripting	Category	Property Type	Description
Selected Location Name	selectedLocationName	Data	String	The name of the selected location.

Name	Scripting	Category	Property Type	Description
Selected Location Path	selectedLocationPath	Data	String	The path of the currently selected location.
Selected Path Without Project	selectedPathWithoutProject	Data	String	The path of the currently selected location without the leading project name.
Selected Location ID	selectedLocationID	Data	int	The ID of the currently selected location.
Display Path	displayPath	Appearance	Boolean	Displays the full path of the location.
Selection Background	selectionBackground	Appearance	Color	The background color of a selected cell in the dropdown list.
Max Table Width	maxTableWidth	Appearance	int	The maximum width

Name	Scripting	Category	Property Type	Description
				allowed for the dropdown table.
Max Table Height	maxTableHeight	Appearance	int	The maximum height allowed for the dropdown table.
Styles	styles	Appearance	Dataset	Contains the component's styles.

focusLost

This event occurs when a component that had the input focus lost it to another component.

Property	Description
source	The component that fired this event.
oppositeComponent	The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vise versa.

key

keyPressed

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

keyReleased

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	

Property	Description
	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

keyTyped

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
altDown	

Property	Description
	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

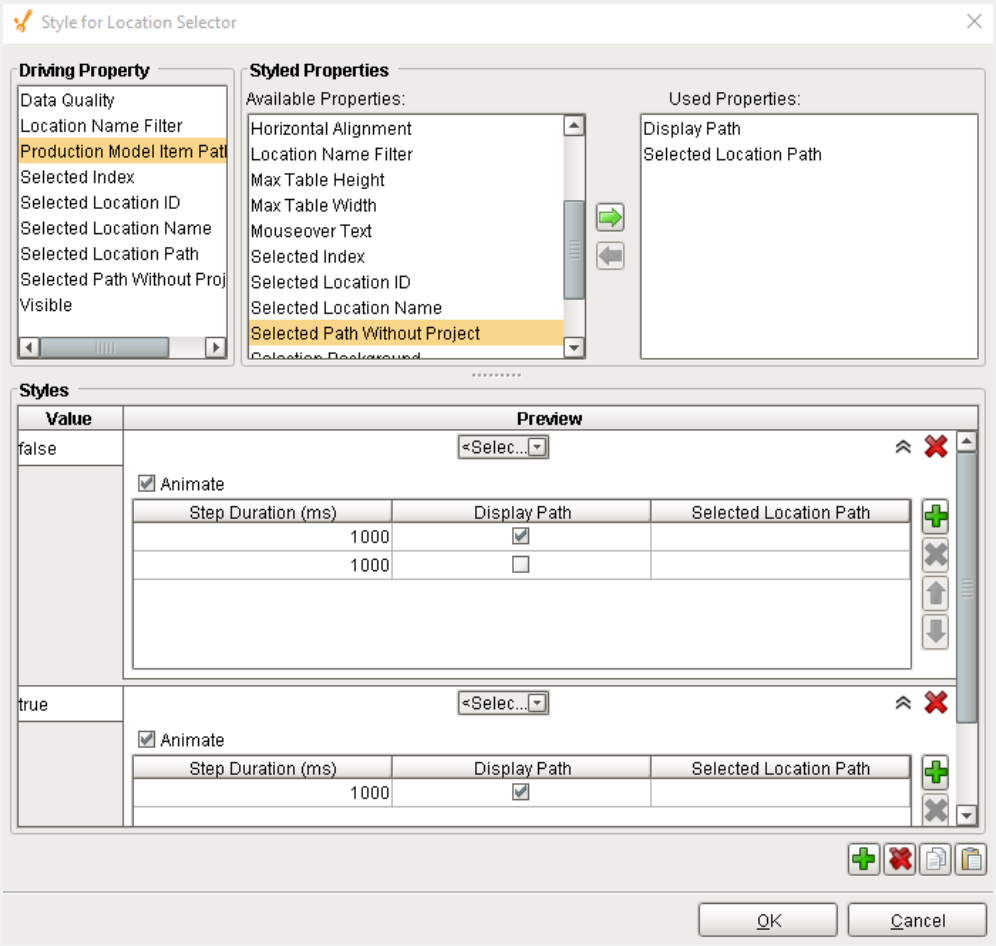
propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

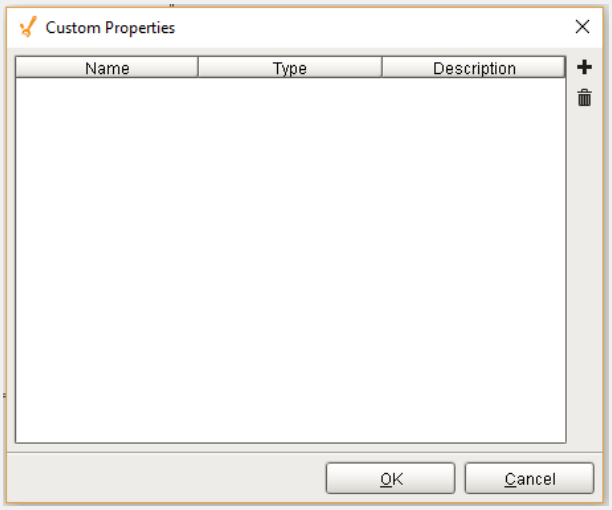
Customizers

This component does not have a customizer however this component relies on custom styles. The example below has the styles defined here:

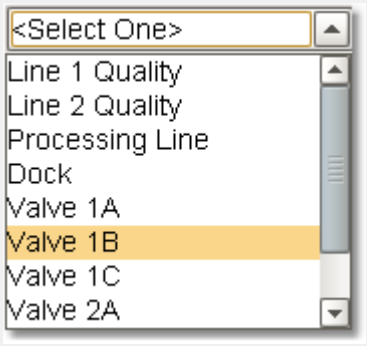


Custom Properties

The custom properties can be used to add user defined properties.



Examples

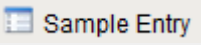


SPCLocationSelector

Location Selector

Sample Entry

General

Component Palette Icon: 

Description

A component used to display and enter sample measurement data. The entry fields are dynamically created based on attributes defined in the sample definition. Additionally, the number of measurements are defined by the measurement count setting in the sample definition. The Up Down Traversal property can be used to change the field tab order between column and row. When saving, the measurement data is validated, and if any validation errors exists a message is displayed to the user.

Properties

Name	Scripting	Category	Property	Description

			Type	
Foreground Color	foregroundColor	Appearance	Color	The foreground color of the component.
Background Color	backgroundColor	Appearance	Color	The background color of the component.
Up Down Traversal	upDownTraversal	Behavior	Boolean	Traverse from top to bottom as values are entered.
Show Units	showUnits	Behavior	Boolean	Shows the units values for each attribute.
Measurement Label	measurementLabel	Appearance	String	Text to display for the measurement label.
Read Only	readOnly	Behavior	Boolean	Prevent entering or changing sample values.
Title	title	Appearance	String	The title of the test.
Title Font	titleFont	Appearance	Font	The font for the title.

Name	Scripting	Category	Property Type	Description
Label Font	labelFont	Appearance	Font	The font for the title.
Measurement Number Font	numberFont	Appearance	Font	The font for the measurement number.
Entry Field Font	fieldFont	Appearance	Font	The font for the data entry fields.
Column Gap size	gapx	Appearance	int	Size of gap between column fields in pixels.
Row Gap size	gapy	Appearance	int	Size of gap between row fields in pixels.
Sample Taken Date Time	sampleTakenDateTime	Data	DateTime	The date and time the sample was taken.
Selected Attribute	selectedAttribute	Data	String	The selected attribute.
Selected Measurement Number	selectedMeasNo	Data	Int4	The selected attribute measurement number.
Sample Taken By	sampleTakenBy	Data	String	

Name	Scripting	Category	Property Type	Description
				The name of the user that is recording the sample.

Scripting

Scripting Functions

getSample()

- Description

Returns an existing sample. (See Sample section for more information).

- Parameters

None

- Return

Sample sample - An existing Sample.

- Scope

Client

getSampleUUID()

- Description

Returns the uuid of existing sample. (See Sample section for more information).

- Parameters

None

- Return

String sampleUUID - The uuid of the existing Sample.

- Scope

Client

save

- Description

Save changes made to the measurement values. This method also records the current product code and reference number for the production location.

- Parameters

None

- Return

String Message of any errors that may have occurred during the save operation.

- Scope

Client

save(productCode, refNo)

- Description

Save changes made to the measurements values along with a product code and reference number specified in the parameters.

- Parameters

String productCode - Product code to record along with the measurement values.

String refNo - Reference number to record along with the measurement values.

- Return

String Message of any errors that may have occurred during the save operation.

- Scope

Client

populateMeasurements(measurementValues)

- Description

Populate the list of measurement values.

- Parameters

Map<String, List<Object>> measurementValues - The list of measurement values to be populated.

- Return

Nothing

- Scope

Client

populateMeasurement(attributeName, measurementNo, value)

- Description

Populate the measurement into the sample entry component.

- Parameters

String attributeName - Name of the attribute to return the measurement value for.

Integer measurementNo - The measurement number associated with the sample.

Object value - The value of the measurement.

- Return

Nothing

- Scope

Client

populateMeasurement(attributeName, value, moveToNextMeasurement)

- Description

Populate the measurement into the sample entry component.

- Parameters

String attributeName - Name of the attribute to return the measurement value for.

Object value - The value of the measurement.

Boolean moveToNextMeasurement - True if the value moved to the next measurement.

- Return

Nothing

- Scope

Client

populateMeasurement(value, moveToNextMeasurement)

- Description

Populate the measurement into the sample entry component.

- Parameters

Object value - The value of the measurement.

Boolean moveToNextMeasurement - True if the value moved to the next measurement.

- Return

Nothing

- Scope

Client

selectMeasurement(attributeName, measurementNo)

- Description

Selects the measurement specified by the attributeName and measurementNo.

- Parameters

String attributeName - Name of the attribute to select the measurement for.

Integer measurementNo - The measurement number associated with the sample.

- Return

Nothing

- Scope

Client

clearMeasurementValues

- Description

Removes all the measurement values.

- Parameters

None

- Return

Nothing

- Scope

Client

undo

- Description

Any changed measurement values will be restored to their original values.

- Parameters

None

- Return

Nothing

- Scope

Client

approve

- Description

Approve the current sample .

- Parameters

None

- Return

String Message of any errors that may have occurred during the approve operation.

- Scope

Client

unapprove

- Description

Unapprove the current sample .

- Parameters

None

- Return

String Message of any errors that may have occurred during the unapprove operation.

- Scope

Client

exclude

- Description

Excludes the sample specified by uuid parameter.

- Parameters

String sampleUUID - The UUID to an existing sample to exclude.

- Return

String Message of any errors that may have occurred during the operation.

- Scope

Client

include

- Description

Includes the sample specified by uuid parameter.

- Parameters

String sampleUUID - The UUID to an existing sample to include.

- Return

String Message of any errors that may have occurred during the operation.

- Scope

Client

showEditNotePopup

- Description

Show the note popup to allow the user to add or edit the note tied to the currently selected sample.

- Parameters

None

- Return

String Message of any errors that may have occurred during the show note operation.

- Scope

Client

validateMeasurementLimits

- Description

Checks if the measurement values are within the spec or control limit.

- Parameters

None

- Return

String Message of any errors that may have occurred during the validation.

- Scope

Client

Extension Functions

measurementSelected

- Description

Called for each measurement when selected. Do not block, sleep, or execute any I/O; called on painting thread.

- Parameters

self - A reference to the component that is invoking this function.

attributeName - The selected attribute name.

measurementNumber - The measurement number selected.

value - The value of this measurement.

attribute - The attribute.

attrLimits - A dataset containing the attribute control and spec limits.

- Return

Nothing

- Scope

Client

measurementValueEntered

- Description

Called for each measurement after a value is entered. Do not block, sleep, or execute any I/O; called on painting thread.

- Parameters

self - A reference to the component that is invoking this function.

attributeName - The selected attribute name.

measurementNumber - The measurement number selected.

value - The value of this measurement.

attribute - The attribute.

isWithinLimits - True if this value is within limits.

attrLimits - A dataset containing the attribute control and spec limits.

- Return

Nothing

- Scope

Client

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

Property	Description
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

Property	Description
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	

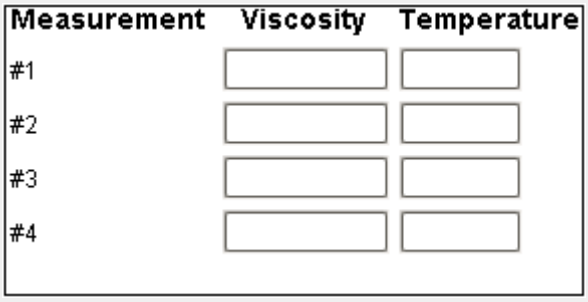
Property	Description
	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

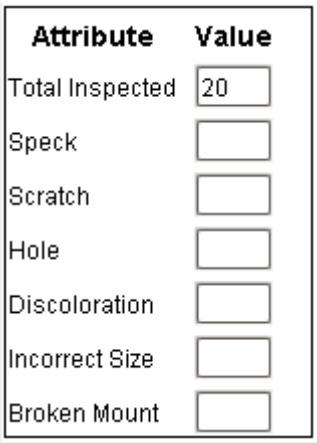
This component does not have any custom properties.

Examples

Depending on the measurement count defined in the sample definition, the orientation of the edit fields will change. If the measurement count is greater than 1, then there will be a row for each measurement with the attributes appearing horizontally. If the measurement count is equal to 1, then the attributes appear vertically in separate rows. This reduces the need for the user to have to scroll while entering sample data if there are a number of attributes.

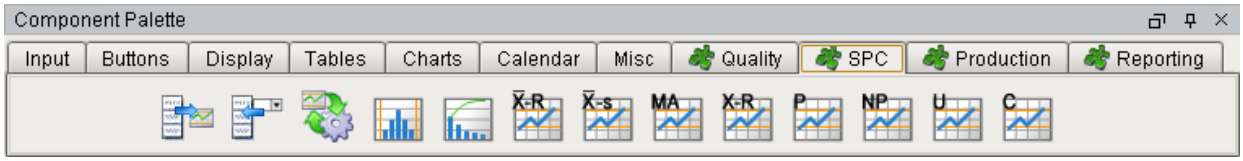


Multiple Measurement Sample Entry



Single Measurement Sample Entry

SPC Component Tab

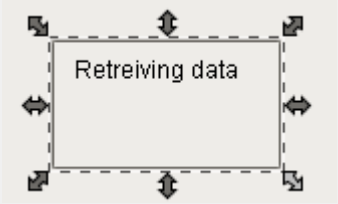


SPCComponents

SPC Components

C-Chart

General



Component Palette Icon:

C-Chart

Description

The Number of Nonconformities (c) control chart is used to display SPC results that have nonconformities counts for each sample. It does not retrieve SPC results from the SPC module so it must be used with either the SPC Selector or the SPC Controller components that do. Only SPC results with c chart SPC Data Format specified will be displayed.

Properties

Name	Scripting	Category	Property Type	Description
No Data Font	noDataFont	Appearance	Font	The font to use for the no data message.
No Data Foreground	noDataForeground	Appearance	Color	The foreground color of the no data message.
No Data Message	noDataMessage	Appearance	String	The message to show when there is no data.
Edit Control Limit Image	editControlLimitImagePath	Chart	String	Image to show for editing control limits.
Enable Control Limit Editing	enableControlLimitEditing	Chart	boolean	If true, allow users to change control limit values.
Enable Note Editing	enableNoteEditing	Chart	boolean	If true, allow users to add and edit notes.
	enablePointDeletion	Chart	boolean	

Name	Scripting	Category	Property Type	Description
Enable Point Deletion				If true, allow users to delete points.
Horizontal Grid Line Color	horzGridLineColor	Chart	Color	The color of horizontal grid lines.
Limit Dialog Horizontal Offset	limitDialogOffsetX	Chart	String	The horizontal offset to display the control limit dialog box.
Limit Dialog Vertical Offset	limitDialogOffsetY	Chart	String	The vertical offset to display the control limit dialog box.
Marker Image Path	markerImagePath	Chart	String	The relative path of an image to display for markers.
Marker Label Font	markerLabelFont	Chart	Font	The font to use for the markers.
Note Image	noteImagePath	Chart	String	Image to show on the the charts when notes exist for a sample.
	primaryChartBackground	Chart	Color	

Name	Scripting	Category	Property Type	Description
Primary Chart Background				The background color of the primary chart.
Right Axis Width	rightAxisWidth	Chart	int	The width of the right chart axis.
Secondary Chart Background	secondaryChartBackground	Chart	Color	The background color of the secondary chart.
Show Horizontal Grid Lines	showHorzGridLines	Chart	boolean	If true, show horizontal grid lines on charts.
Show Notes	showNotes	Chart	boolean	If true, show notes on the chart.
Show Primary Average	showPrimaryAverage	Chart	boolean	Set to true to display the average line on the primary chart.
Show Primary Chart	showPrimaryChart	Chart	boolean	Set to true to display the primary chart.
Show Secondary Average	showSecondaryAverage	Chart	boolean	Set to true to display the average line

Name	Scripting	Category	Property Type	Description
				on the secondary chart.
Show Seconday Chart	showSecondaryChart	Chart	boolean	Set to true to display the secondary chart.
Show Vertical Grid Lines	showVertGridLines	Chart	boolean	If true, show vertical grid lines on charts.
Vertical Grid Line Color	vertGridLineColor	Chart	Color	The color of vertical grid lines.
Calculated Values	calcValues	Data	String	Calculated value definitions.
Measurement Count	measurementCount	Data	int	Number of measurements in the SPC results.
SPC Data	sPCData	Data	String	SPC Data.
SPC Results	sPCResults	Data	String	SPC results containing data, measurement count and calculated value information.

Name	Scripting	Category	Property Type	Description
User	user	Data	String	Current user. If this property is not set, then the currently logged in user will be used.
Calc Background	calcBackground	Table	Color	The background color of the calculated values.
CalcFont	calcFont	Table	Font	The font to use for the calculated values.
Calc Foreground	calcForeground	Table	Color	The foreground color of the calculated values.
Column Width	columnWidth	Table	int	The width of the table columns.
Data Background	dataBackground	Table	Color	The background color of the data values.
Data Font	dataFont	Table	Font	The font to use for the data values.

Name	Scripting	Category	Property Type	Description
Data Foreground	dataForeground	Table	Color	The foreground color of the data values.
Date Background	dateBackground	Table	Color	The background color of the date row.
Date Font	dateFont	Table	Font	The font to use for the date row.
Date Foreground	dateForeground	Table	Color	The foreground color of the date row.
Date Format	dateFormat	Table	String	The date formatting pattern used to display the date.
Enable Highlights	enableHighlights	Table	String	Enables highlighting of signals and control limits in the table.
Label Background	labelBackground	Table	Color	The background color of the labels.
Label Font	labelFont	Table	Font	

Name	Scripting	Category	Property Type	Description
				The font to use for the labels.
Label Foreground	labelForeground	Table	Color	The foreground color of the labels.
Min Visible Samples	minVisibleSamples	Table	String	The minimum number of sample to show in the table.
Row Height	rowHeight	Table	int	The height of the table rows.
Scroll X	scrollX	Table	String	The scroll bar x poition.
Show Table	showTable	Table	boolean	Set to true to display the sample values table.
Visible Measurements	visibleMeasurements	Table	String	The number of measurements to show in the table.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

getNoteToolTip

- Description

Called when tool tip content is generated for a note symbol. The default tool tip is passed and can be modified or replaced.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample that the tool tip is being generated for.

defaultToolTip - The default tool tip that can be modified or replaced.

- Return

The defaultToolTip.

- Scope

Client

getSampleToolTip

- Description

Called when tool tip content is generated for a sample. The default tool tip is passed and can be modified or replaced.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample that the tool tip is being generated for.

valueInfo - The SPCCalcValueInfo object that holds information about the value.

defaultToolTip - The default tool tip that can be modified or replaced.

- Return

The defaultToolTip.

- Scope

Client

beforeSampleDelete

- Description

Called before the sample be deleted. If return value is True, then the sample be deleted.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample to be deleted.

- Return

True

- Scope

Client

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.

Property	Description
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description	
source	The component that fired this event.	
button	The code for the button that caused this event to fire.	
clickCount	The number of mouse clicks associated with this event.	
x	The x-coordinate (with respect to the source component) of this mouse event.	
y	The y-coordinate (with respect to the source component) of this mouse event.	
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.	
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.	
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.	
shiftDown		

Property	Description	
	True (1) if the Shift key was held down during this event, false (0) otherwise.	

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

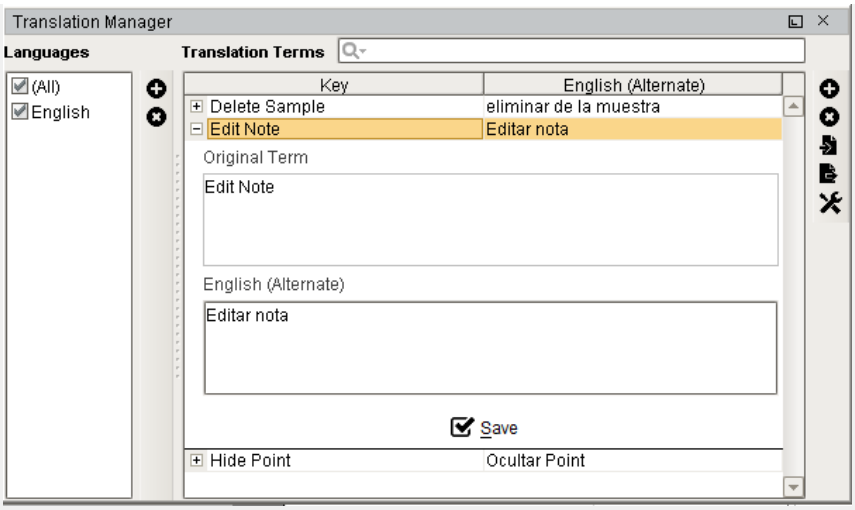
propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Control Chart Menu Items

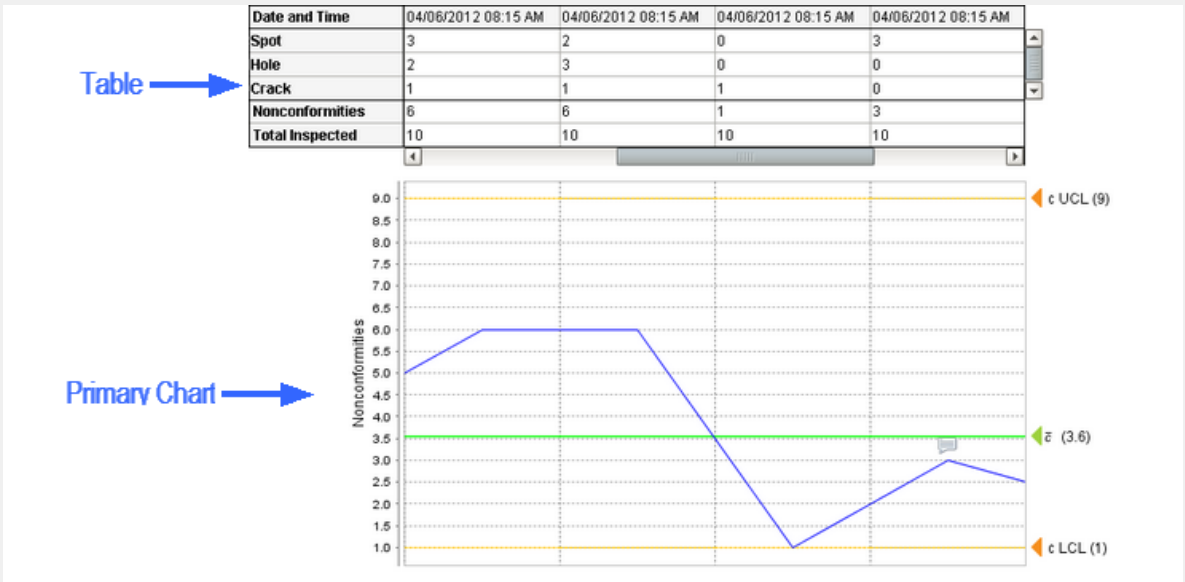
Right clicking the control chart will give the pop up menu items like Delete Sample, Edit Note, Hide Point and Restore Hidden Points. SPC chart popup panel menu strings are localizable. The alternate strings can be added through the Ignition translation manager (not the component translation manager). Reopen the control chart page and right click on a point to manifest the changes.



Customizers

This component does not have any custom properties.

Examples

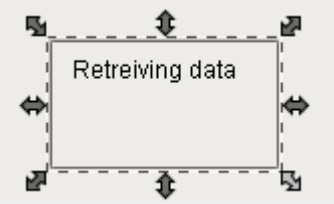


SPCCChart

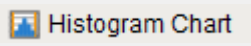
C Control Chart

Histogram Chart

General



Component Palette Icon:



Description

The Histogram chart is used to display frequency distribution of sample measurements. It does not retrieve SPC results from the SPC module so it must be used with either the SPC Selector or the SPC Controller components that do. Only SPC results with Histogram SPC Data Format specified will be displayed.

Properties

Through the use of the properties listed below, the appearance and functionality of this component can be modified as desired.

Name	Scripting	Category	Property Type	Description
Vertical	vertical	Chart	Boolean	The orientation of the chart.
Chart Background Color	chartBackground	Chart	Color	The color of the chart background.
Bar Color	barColor	Chart	Color	

Name	Scripting	Category	Property Type	Description
				The color of the bars for the chart.
Bar Spacing	barSpacing	Chart	Float8	Spacing between bar as a percentage of the bar width.
Gradient	gradient	Chart	Boolean	If true, bars will be painted with a gradient 'shine'.
Shadows	shadow	Chart	Boolean	If true, show shadows for bars.
Tick Label Font	tickLabelFont	Chart	Font	The font of the tick labels.
Tick Label Color	tickLabelColor	Chart	Color	The color of the tick labels.
Value Axis Title	valueAxisTitle	Chart	String	Title to show for the value axis.
Frequency Axis Title	frequencyAxisTitle	Chart	String	Title to show for the frequency axis.
Axis Title Font	axisTitleFont	Chart	Font	Font for both axis titles.
Axis Title Color	axisTitleColor	Chart	Color	Color to display both axis titles.
Vertical Grid Line Color	vertGridLineColor	Chart	Color	The color of vertical grid lines.
	showVertGridLines	Chart	Boolean	

Name	Scripting	Category	Property Type	Description
Show Vertical Grid Lines				If true, show vertical grid lines on charts.
Horizontal Grid Line Color	horzGridLineColor	Chart	Color	The color of horizontal grid lines.
Show Horizontal Grid Lines	showHorzGridLines	Chart	Boolean	If true, show horizontal grid lines on charts.

Scripting

Scripting Functions

showSetLimitPanel

- Description

Causes the calculate and set control limit dialog to be shown.

- Parameters

None

- Return

Nothing

- Scope

Client

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	

Property	Description
	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

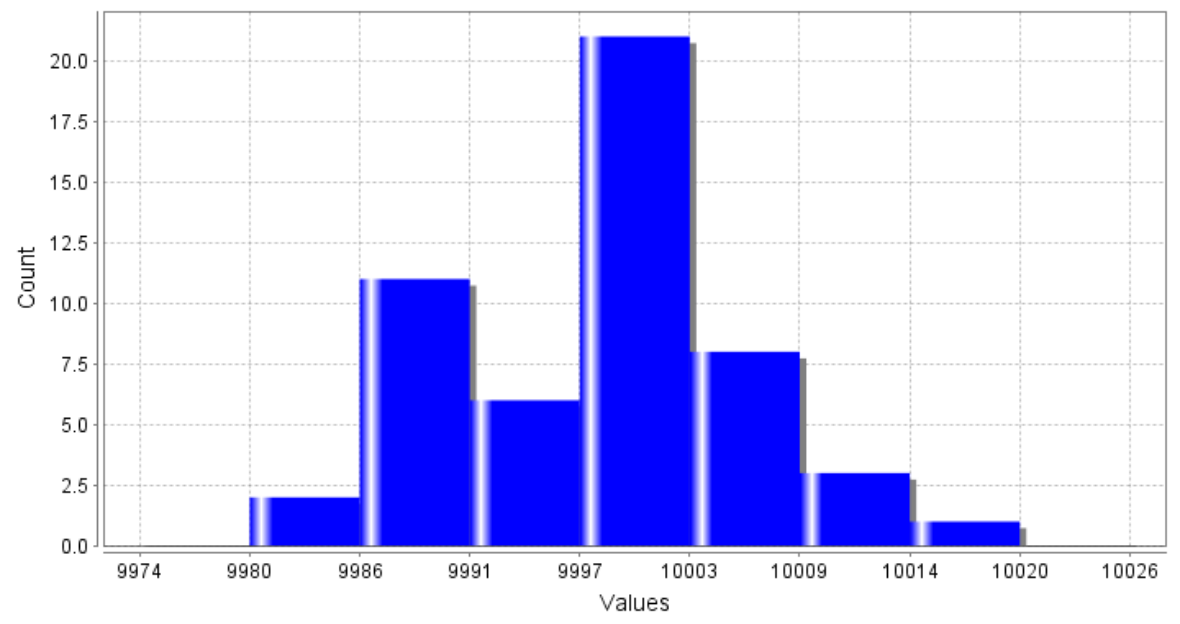
Property	Description
source	The component that fired this event.

Property	Description
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

This component does not have any custom properties.

Examples

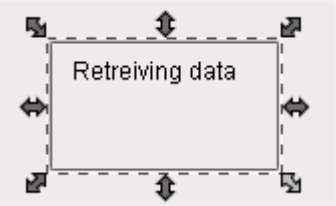


SPCHistogram

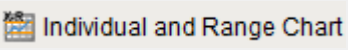
Histogram Chart

Individual and Range Chart

General



Component Palette Icon:



Description

The Individual Moving Range (MR) control chart is used to display SPC results that have a single measurement for each sample. It does not retrieve SPC results from the SPC module so it must be used with either the SPC Selector or the SPC Controller components that do. Only SPC results with Individual and MR SPC Data Format specified will be displayed.

Properties

Name	Scripting	Category	Property Type	Description
No Data Font	noDataFont	Appearance	Font	The font to use for the no data message.
No Data Foreground	noDataForeground	Appearance	Color	The foreground color of the no data message.

Name	Scripting	Category	Property Type	Description
No Data Message	noDataMessage	Appearance	String	The message to show when there is no data.
Edit Control Limit Image	editControlLimitImagePath	Chart	String	Image to show for editing control limits.
Enable Control Limit Editing	enableControlLimitEditing	Chart	boolean	If true, allow users to change control limit values.
Enable Note Editing	enableNoteEditing	Chart	boolean	If true, allow users to add and edit notes.
Enable Point Deletion	enablePointDeletion	Chart	boolean	If true, allow users to delete points.
Horizontal Grid Line Color	horzGridLineColor	Chart	Color	The color of horizontal grid lines.
Limit Dialog Horizontal Offset	limitDialogOffsetX	Chart	String	The horizontal offset to display the control limit dialog box.
Limit Dialog Vertical Offset	limitDialogOffsetY	Chart	String	The vertical offset to display the control limit dialog box.

Name	Scripting	Category	Property Type	Description
Marker Image Path	markerImagePath	Chart	String	The relative path of an image to display for markers.
Marker Label Font	markerLabelFont	Chart	Font	The font to use for the markers.
Note Image	noteImagePath	Chart	String	Image to show on the the charts when notes exist for a sample.
Primary Chart Background	primaryChartBackground	Chart	Color	The background color of the primary chart.
Right Axis Width	rightAxisWidth	Chart	int	The width of the right chart axis.
Secondary Chart Background	secondaryChartBackground	Chart	Color	The background color of the secondary chart.
Show Horizontal Grid Lines	showHorzGridLines	Chart	boolean	If true, show horizontal grid lines on charts.
Show Notes	showNotes	Chart	boolean	

Name	Scripting	Category	Property Type	Description
				If true, show notes on the chart.
Show Primary Average	showPrimaryAverage	Chart	boolean	Set to true to display the average line on the primary chart.
Show Primary Chart	showPrimaryChart	Chart	boolean	Set to true to display the primary chart.
Show Secondary Average	showSecondaryAverage	Chart	boolean	Set to true to display the average line on the secondary chart.
Show Seconday Chart	showSecondaryChart	Chart	boolean	Set to true to display the secondary chart.
Show Vertical Grid Lines	showVertGridLines	Chart	boolean	If true, show vertical grid lines on charts.
Vertical Grid Line Color	vertGridLineColor	Chart	Color	The color of vertical grid lines.
Calculated Values	calcValues	Data	String	

Name	Scripting	Category	Property Type	Description
				Calculated value definitions.
Measurement Count	measurementCount	Data	int	Number of measurements in the SPC results.
SPC Data	sPCData	Data	String	SPC Data
SPC Results	sPCResults	Data	String	SPC results containing data, measurement count and calculated value information.
User	user	Data	String	Current user. If this property is not set, then the currently logged in user will be used.
Calc Background	calcBackground	Table	Color	The background color of the calculated values.
CalcFont	calcFont	Table	Font	The font to use for the calculated values.

Name	Scripting	Category	Property Type	Description
Calc Foreground	calcForeground	Table	Color	The foreground color of the calculated values.
Column Width	columnWidth	Table	int	The width of the table columns.
Data Background	dataBackground	Table	Color	The background color of the data values.
Data Font	dataFont	Table	Font	The font to use for the data values.
Data Foreground	dataForeground	Table	Color	The foreground color of the data values.
Date Background	dateBackground	Table	Color	The background color of the date row.
Date Font	dateFont	Table	Font	The font to use for the date row.
Date Foreground	dateForeground	Table	Color	The foreground color of the date row.

Name	Scripting	Category	Property Type	Description
Date Format	dateFormat	Table	String	The date formatting pattern used to display the date.
Enable Highlights	enableHighlights	Table	String	Enables highlighting of signals and control limits in the table.
Label Background	labelBackground	Table	Color	The background color of the labels.
Label Font	labelFont	Table	Font	The font to use for the labels.
Label Foreground	labelForeground	Table	Color	The foreground color of the labels.
Min Visible Samples	minVisibleSamples	Table	String	The minimum number of sample to show in the table.
Row Height	rowHeight	Table	int	The height of the table rows.
Scroll X	scrollX	Table	String	The scroll bar x poition.

Name	Scripting	Category	Property Type	Description
Show Table	showTable	Table	boolean	Set to true to display the sample values table.
Visible Measurements	visibleMeasurements	Table	String	The number of measurements to show in the table.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

getNoteToolTip

- Description

Called when tool tip content is generated for a note symbol. The default tool tip is passed and can be modified or replaced.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample that the tool tip is being generated for.

defaultToolTip - The default tool tip that can be modified or replaced.

- Return

The defaultToolTip.

- Scope

Client

getSampleToolTip

- Description

Called when tool tip content is generated for a sample. The default tool tip is passed and can be modified or replaced.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample that the tool tip is being generated for.

valueInfo - The SPCCalcValueInfo object that holds information about the value.

defaultToolTip - The default tool tip that can be modified or replaced.

- Return

The defaultToolTip.

- Scope

Client

beforeSampleDelete

- Description

Called before the sample be deleted. If return value is True, then the sample be deleted.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample to be deleted.

- Return

True

- Scope

Client

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	**Description**
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

Property	Description
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

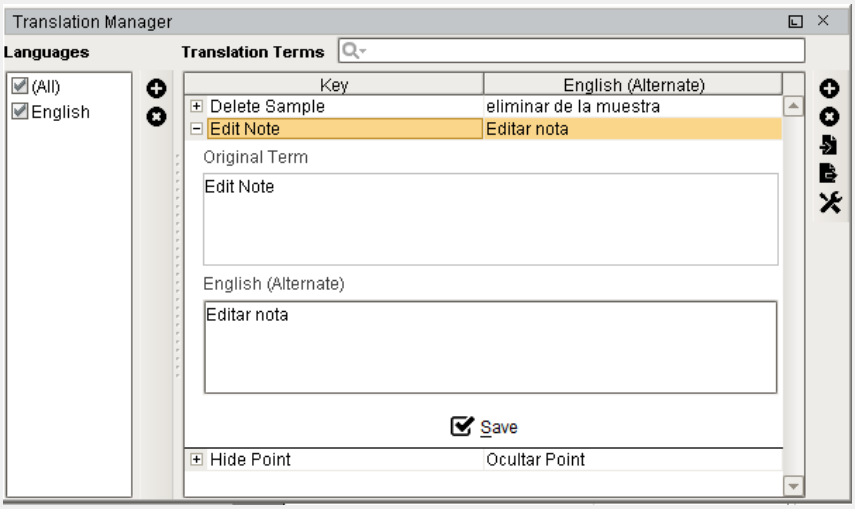
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	

Property	Description
	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Control Chart Menu Items

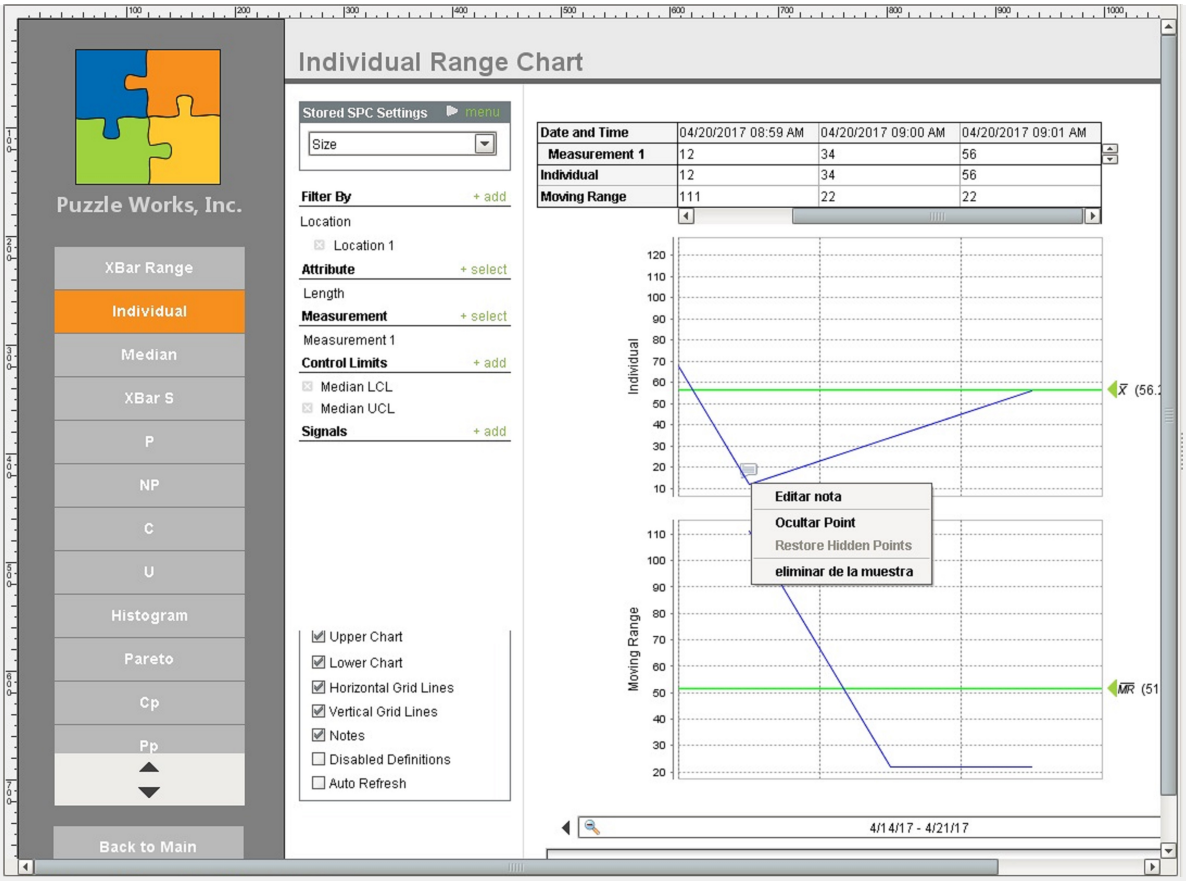
Right clicking the control chart will give the pop up menu items like Delete Sample, Edit Note, Hide Point and Restore Hidden Points. SPC chart popup panel menu strings are localizable. The alternate strings can be added through the Ignition translation manager (not the component translation manager). Reopen the control chart page and right click on a point to manifest the changes.



Customizers

This component does not have any custom properties.

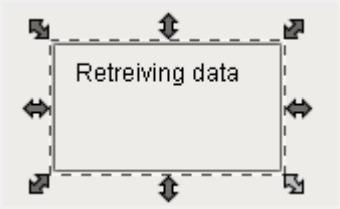
Examples



Individual Moving Range Control Chart

Median and Range Chart

General



Component Palette Icon: Median and Range Chart

Description

The Median Moving Range (MR) control chart is used to display SPC results that have multiple measurements for each sample. It does not retrieve SPC results from the SPC module so it must be used with either the SPC Selector or the SPC Controller components that do. Only SPC results with Median and MR SPC Data Format specified will be displayed.

Properties

Name	Scripting	Category	Property Type	Description
No Data Font	noDataFont	Appearance	Font	The font to use for the no data message.
No Data Foreground	noDataForeground	Appearance	Color	The foreground color of the no data message.
No Data Message	noDataMessage	Appearance	String	The message to show when there is no data.
Edit Control Limit Image	editControlLimitImagePath	Chart	String	Image to show for editing control limits.
Enable Control Limit Editing	enableControlLimitEditing	Chart	boolean	If true, allow users to change control limit values.
Enable Note Editing	enableNoteEditing	Chart	boolean	If true, allow users to add and edit notes.

Name	Scripting	Category	Property Type	Description
Enable Point Deletion	enablePointDeletion	Chart	boolean	If true, allow users to delete points.
Horizontal Grid Line Color	horzGridLineColor	Chart	Color	The color of horizontal grid lines.
Limit Dialog Horizontal Offset	limitDialogOffsetX	Chart	String	The horizontal offset to display the control limit dialog box.
Limit Dialog Vertical Offset	limitDialogOffsetY	Chart	String	The vertical offset to display the control limit dialog box.
Marker Image Path	markerImagePath	Chart	String	The relative path of an image to display for markers.
Marker Label Font	markerLabelFont	Chart	Font	The font to use for the markers.
Note Image	noteImagePath	Chart	String	Image to show on the the charts when notes exist for a sample.
	primaryChartBackground	Chart	Color	

Name	Scripting	Category	Property Type	Description
Primary Chart Background				The background color of the primary chart.
Right Axis Width	rightAxisWidth	Chart	int	The width of the right chart axis.
Secondary Chart Background	secondaryChartBackground	Chart	Color	The background color of the secondary chart.
Show Horizontal Grid Lines	showHorzGridLines	Chart	boolean	If true, show horizontal grid lines on charts.
Show Notes	showNotes	Chart	boolean	If true, show notes on the chart.
Show Primary Average	showPrimaryAverage	Chart	boolean	Set to true to display the average line on the primary chart.
Show Primary Chart	showPrimaryChart	Chart	boolean	Set to true to display the primary chart.
Show Secondary Average	showSecondaryAverage	Chart	boolean	Set to true to display the average line

Name	Scripting	Category	Property Type	Description
				on the secondary chart.
Show Seconday Chart	showSecondaryChart	Chart	boolean	Set to true to display the secondary chart.
Show Vertical Grid Lines	showVertGridLines	Chart	boolean	If true, show vertical grid lines on charts.
Vertical Grid Line Color	vertGridLineColor	Chart	Color	The color of vertical grid lines.
Calculated Values	calcValues	Data	String	Calculated value definitions.
Measurement Count	measurementCount	Data	int	Number of measurements in the SPC results.
SPC Data	sPCData	Data	String	SPC Data.
SPC Results	sPCResults	Data	String	SPC results containing data, measurement count and calculated value information.

Name	Scripting	Category	Property Type	Description
User	user	Data	String	Current user. If this property is not set, then the currently logged in user will be used.
Calc Background	calcBackground	Table	Color	The background color of the calculated values.
CalcFont	calcFont	Table	Font	The font to use for the calculated values.
Calc Foreground	calcForeground	Table	Color	The foreground color of the calculated values.
Column Width	columnWidth	Table	int	The width of the table columns.
Data Background	dataBackground	Table	Color	The background color of the data values.
Data Font	dataFont	Table	Font	The font to use for the data values.

Name	Scripting	Category	Property Type	Description
Data Foreground	dataForeground	Table	Color	The foreground color of the data values.
Date Background	dateBackground	Table	Color	The background color of the date row.
Date Font	dateFont	Table	Font	The font to use for the date row.
Date Foreground	dateForeground	Table	Color	The foreground color of the date row.
Date Format	dateFormat	Table	String	The date formatting pattern used to display the date.
Enable Highlights	enableHighlights	Table	String	Enables highlighting of signals and control limits in the table.
Label Background	labelBackground	Table	Color	The background color of the labels.
Label Font	labelFont	Table	Font	

Name	Scripting	Category	Property Type	Description
				The font to use for the labels.
Label Foreground	labelForeground	Table	Color	The foreground color of the labels.
Min Visible Samples	minVisibleSamples	Table	String	The minimum number of sample to show in the table.
Row Height	rowHeight	Table	int	The height of the table rows.
Scroll X	scrollX	Table	String	The scroll bar x poition.
Show Table	showTable	Table	boolean	Set to true to display the sample values table.
Visible Measurements	visibleMeasurements	Table	String	The number of measurements to show in the table.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

getNoteToolTip

- Description

Called when tool tip content is generated for a note symbol. The default tool tip is passed and can be modified or replaced.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample that the tool tip is being generated for.

defaultToolTip - The default tool tip that can be modified or replaced.

- Return

The defaultToolTip.

- Scope

Client

getSampleToolTip

- Description

Called when tool tip content is generated for a sample. The default tool tip is passed and can be modified or replaced.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample that the tool tip is being generated for.

valueInfo - The SPCCalcValueInfo object that holds information about the value.

defaultToolTip - The default tool tip that can be modified or replaced.

- Return

The defaultToolTip.

- Scope

Client

beforeSampleDelete

- Description

Called before the sample be deleted. If return value is True, then the sample be deleted.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample to be deleted.

- Return

True

- Scope

Client

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.

Property	Description
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	

Property	Description
	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

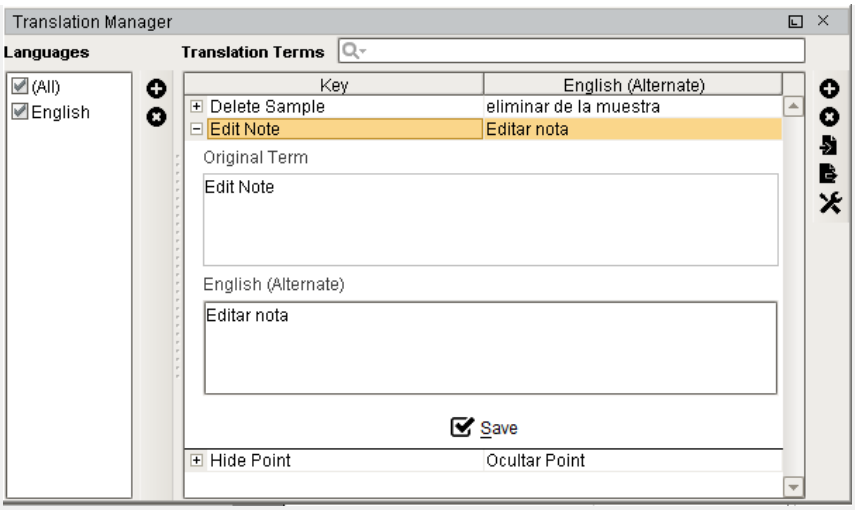
propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Control Chart Menu Items

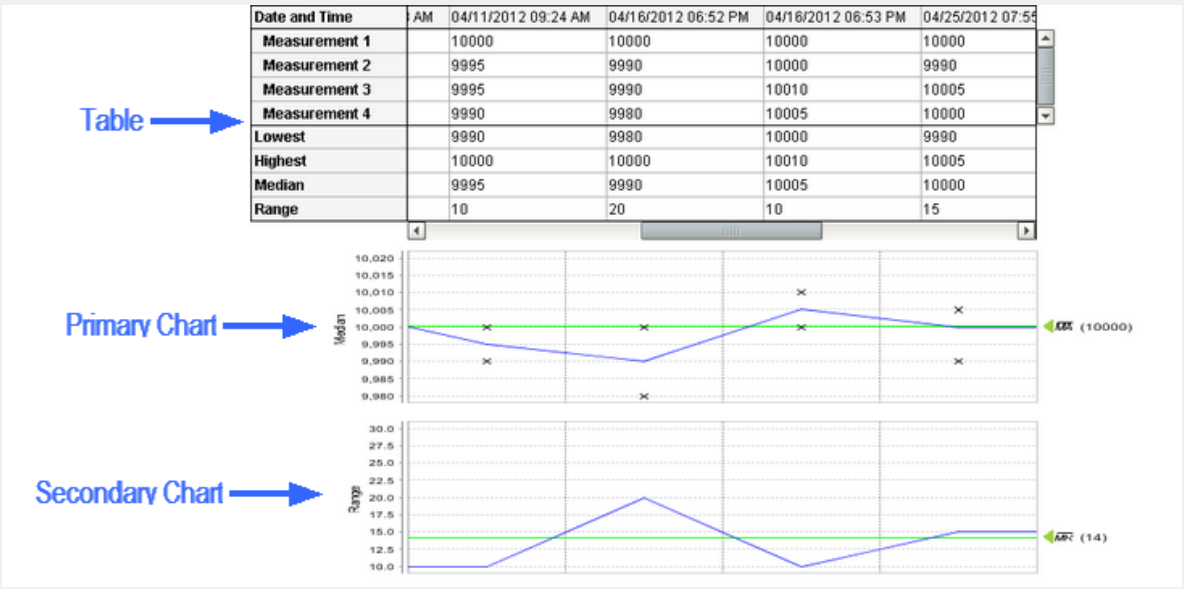
Right clicking the control chart will give the pop up menu items like Delete Sample, Edit Note, Hide Point and Restore Hidden Points. SPC chart popup panel menu strings are localizable. The alternate strings can be added through the Ignition translation manager (not the component translation manager). Reopen the control chart page and right click on a point to manifest the changes.



Customizers

This component does not have any custom properties.

Examples

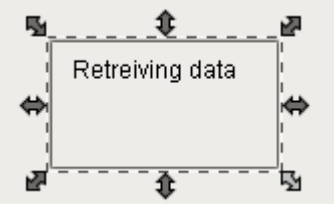


SPCMedian

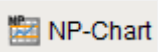
Median Moving Range Control Chart

NP-Chart

General



Component Palette Icon:



Description

The Number of Nonconforming Items (np) control chart is used to display SPC results that have nonconforming counts for each sample. It does not retrieve SPC results from the SPC module so it must be used with either the SPC Selector or the SPC Controller components that do. Only SPC results with np chart SPC Data Format specified will be displayed.

Properties

Name	Scripting	Category	Property Type	Description
No Data Font	noDataFont	Appearance	Font	The font to use for the no data message.
No Data Foreground	noDataForeground	Appearance	Color	The foreground color of the no data message.
	noDataMessage	Appearance	String	

Name	Scripting	Category	Property Type	Description
No Data Message				The message to show when there is no data.
Edit Control Limit Image	editControlLimitImagePath	Chart	String	Image to show for editing control limits.
Enable Control Limit Editing	enableControlLimitEditing	Chart	boolean	If true, allow users to change control limit values.
Enable Note Editing	enableNoteEditing	Chart	boolean	If true, allow users to add and edit notes.
Enable Point Deletion	enablePointDeletion	Chart	boolean	If true, allow users to delete points.
Horizontal Grid Line Color	horzGridLineColor	Chart	Color	The color of horizontal grid lines.
Limit Dialog Horizontal Offset	limitDialogOffsetX	Chart	String	The horizontal offset to display the control limit dialog box.
Limit Dialog Vertical Offset	limitDialogOffsetY	Chart	String	The vertical offset to display the control limit dialog box.

Name	Scripting	Category	Property Type	Description
Marker Image Path	markerImagePath	Chart	String	The relative path of an image to display for markers.
Marker Label Font	markerLabelFont	Chart	Font	The font to use for the markers.
Note Image	noteImagePath	Chart	String	Image to show on the the charts when notes exist for a sample.
Primary Chart Background	primaryChartBackground	Chart	Color	The background color of the primary chart.
Right Axis Width	rightAxisWidth	Chart	int	The width of the right chart axis.
Secondary Chart Background	secondaryChartBackground	Chart	Color	The background color of the secondary chart.
Show Horizontal Grid Lines	showHorzGridLines	Chart	boolean	If true, show horizontal grid lines on charts.
Show Notes	showNotes	Chart	boolean	

Name	Scripting	Category	Property Type	Description
				If true, show notes on the chart.
Show Primary Average	showPrimaryAverage	Chart	boolean	Set to true to display the average line on the primary chart.
Show Primary Chart	showPrimaryChart	Chart	boolean	Set to true to display the primary chart.
Show Secondary Average	showSecondaryAverage	Chart	boolean	Set to true to display the average line on the secondary chart.
Show Seconday Chart	showSecondaryChart	Chart	boolean	Set to true to display the secondary chart.
Show Vertical Grid Lines	showVertGridLines	Chart	boolean	If true, show vertical grid lines on charts.
Vertical Grid Line Color	vertGridLineColor	Chart	Color	The color of vertical grid lines.
Calculated Values	calcValues	Data	String	

Name	Scripting	Category	Property Type	Description
				Calculated value definitions.
Measurement Count	measurementCount	Data	int	Number of measurements in the SPC results.
SPC Data	sPCData	Data	String	SPC Data.
SPC Results	sPCResults	Data	String	SPC results containing data, measurement count and calculated value information.
User	user	Data	String	Current user. If this property is not set, then the currently logged in user will be used.
Calc Background	calcBackground	Table	Color	The background color of the calculated values.
CalcFont	calcFont	Table	Font	The font to use for the calculated values.

Name	Scripting	Category	Property Type	Description
Calc Foreground	calcForeground	Table	Color	The foreground color of the calculated values.
Column Width	columnWidth	Table	int	The width of the table columns.
Data Background	dataBackground	Table	Color	The background color of the data values.
Data Font	dataFont	Table	Font	The font to use for the data values.
Data Foreground	dataForeground	Table	Color	The foreground color of the data values.
Date Background	dateBackground	Table	Color	The background color of the date row.
Date Font	dateFont	Table	Font	The font to use for the date row.
Date Foreground	dateForeground	Table	Color	The foreground color of the date row.

Name	Scripting	Category	Property Type	Description
Date Format	dateFormat	Table	String	The date formatting pattern used to display the date.
Enable Highlights	enableHighlights	Table	String	Enables highlighting of signals and control limits in the table.
Label Background	labelBackground	Table	Color	The background color of the labels.
Label Font	labelFont	Table	Font	The font to use for the labels.
Label Foreground	labelForeground	Table	Color	The foreground color of the labels.
Min Visible Samples	minVisibleSamples	Table	String	The minimum number of sample to show in the table.
Row Height	rowHeight	Table	int	The height of the table rows.
Scroll X	scrollX	Table	String	The scroll bar x poition.

Name	Scripting	Category	Property Type	Description
Show Table	showTable	Table	boolean	Set to true to display the sample values table.
Visible Measurements	visibleMeasurements	Table	String	The number of measurements to show in the table.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

getNoteToolTip

- Description

Called when tool tip content is generated for a note symbol. The default tool tip is passed and can be modified or replaced.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample that the tool tip is being generated for.

defaultToolTip - The default tool tip that can be modified or replaced.

- Return

The defaultToolTip.

- Scope

Client

getSampleToolTip

- Description

Called when tool tip content is generated for a sample. The default tool tip is passed and can be modified or replaced.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample that the tool tip is being generated for.

valueInfo - The SPCCalcValueInfo object that holds information about the value.

defaultToolTip - The default tool tip that can be modified or replaced.

- Return

The defaultToolTip.

- Scope

Client

beforeSampleDelete

- Description

Called before the sample be deleted. If return value is True, then the sample be deleted.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample to be deleted.

- Return

The defaultToolTip.

- Scope

Client

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	**Description**
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

Property	Description
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

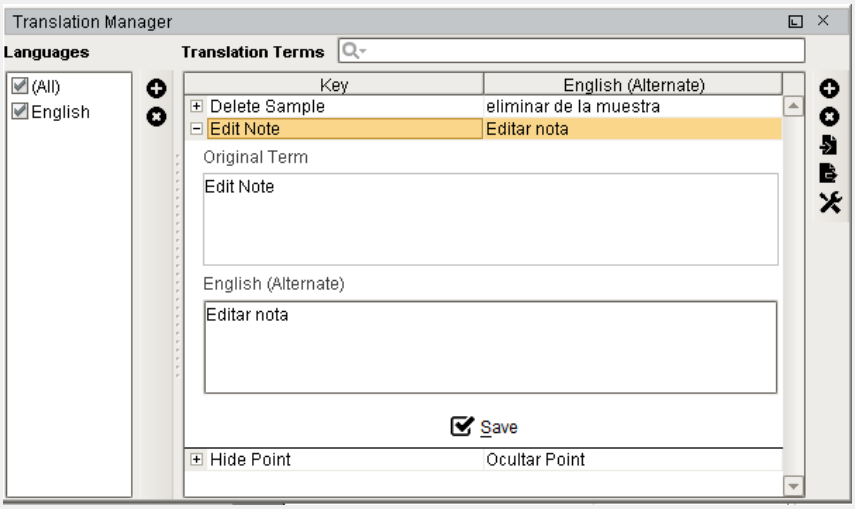
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	

Property	Description
	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Control Chart Menu Items

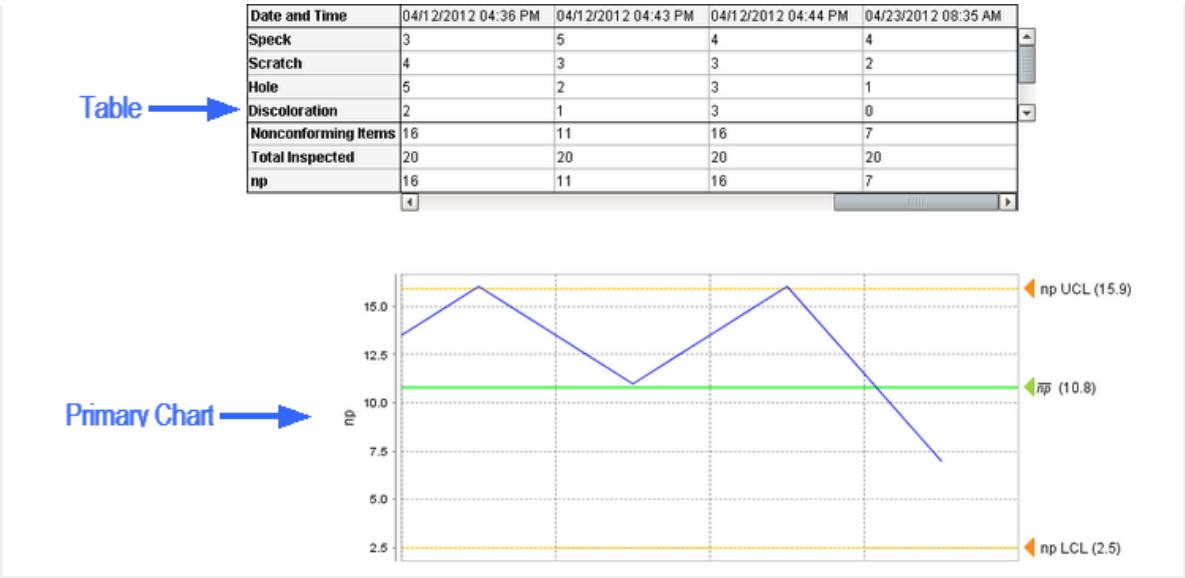
Right clicking the control chart will give the pop up menu items like Delete Sample, Edit Note, Hide Point and Restore Hidden Points. SPC chart popup panel menu strings are localizable. The alternate strings can be added through the Ignition translation manager (not the component translation manager). Reopen the control chart page and right click on a point to manifest the changes.



Customizers

This component does not have any custom properties.

Examples

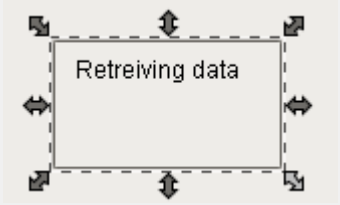


SPCNPChart

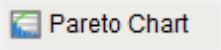
NP Control Chart

Pareto Chart

General



Component Palette Icon:



Description

The Pareto chart is used to display which nonconforming items or nonconformities are the largest issue. It does not retrieve SPC results from the SPC module so it must be used with either the SPC Selector or the SPC Controller components that do. Only SPC results with Pareto SPC Data Format specified will be displayed.

Properties

Name	Scripting	Category	Property Type	Description
Vertical	vertical	Chart	Boolean	The orientation of the chart.
Chart Background Color	chartBackground	Chart	Color	The color of the chart background.
Bar Color	barColor	Chart	Color	The color of the bars for the chart.
Accumulation Line Color	accumulationLineColor	Chart	Color	The color of the accumulation line for the chart.
Bar Spacing	barSpacing	Chart	Float8	Spacing between bar as a percentage of the bar width.
Gradient	gradient	Chart	Boolean	If true, bars will be painted with a gradient 'shine'.
Shadows	shadow	Chart	Boolean	If true, show shadows for bars.
Tick Label Font	tickLabelFont	Chart	Font	The font of the tick labels.
Tick Label Color	tickLabelColor	Chart	Color	The color of the tick labels.
Category Axis Title	categoryAxisTitle	Chart	String	Title to show for the category axis.
	frequencyAxisTitle	Chart	String	

Name	Scripting	Category	Property Type	Description
Frequency Axis Title				Title to show for the frequency axis.
Percent Axis Title	percentAxisTitle	Chart	String	Title to show for the percentage axis.
Axis Title Font	axisTitleFont	Chart	Font	Font for both axis titles.
Axis Title Color	axisTitleColor	Chart	Color	Color to display both axis titles.
Horizontal Grid Line Color	horzGridLineColor	Chart	Color	The color of horizontal grid lines.
Show Horizontal Grid Lines	showHorzGridLines	Chart	Boolean	If true, show horizontal grid lines on charts.
Show Accumulation Line	showAccumulationLine	Chart	Boolean	If true, show the accumulation line on charts.
Maximum Bars To Show	maxBarCount	Chart	int	Limits the number of bars to display on the chart.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

getWeightedValue

- Description

Called to update a category weighted value to use instead of the raw frequency (count). The Show Weighted Values property must be set to True for this extension function to be called.

- Parameters

self - A reference to the component that is invoking this function.

category - The name of the nonconforming or nonconformity attribute.

weight - The weight value defined in the attribute. If a weight value has not been defined for the attribute, then the default of 1.0 is used.

frequency - The frequency (count) value for the category before it is adjusted.

- Return

The weighted value (frequency * weight).

- Scope

Client

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	

Property	Description
	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

Property	Description
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

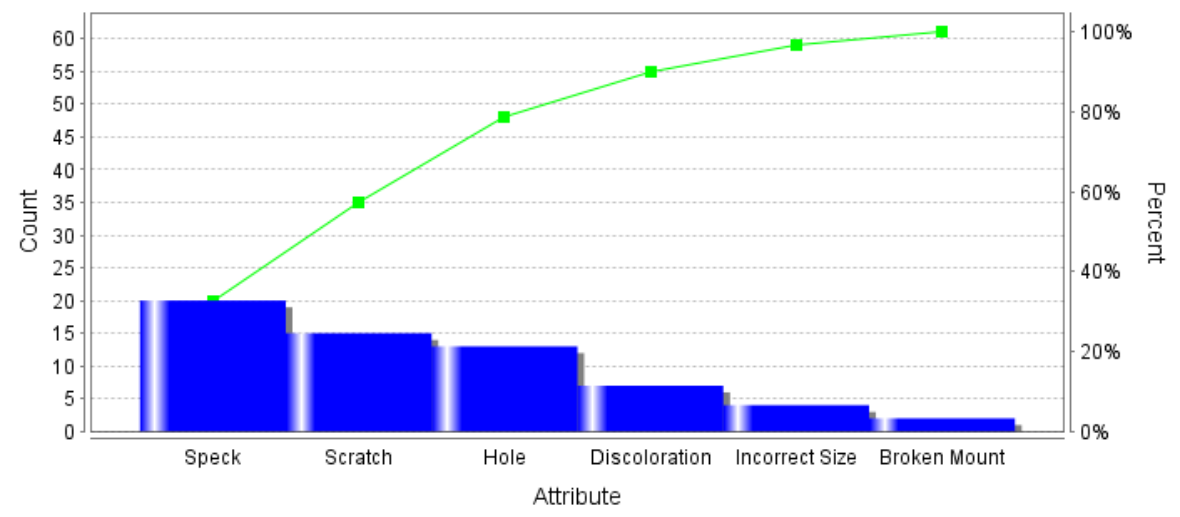
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

This component does not have any custom properties.

Examples

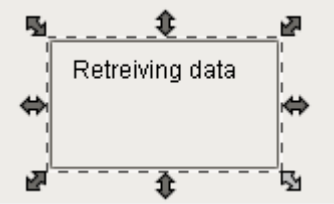


SPCPareto

Pareto Chart

P-Chart

General



Component Palette Icon: P-Chart

Description

The Percentage of Nonconforming Items (p) control chart is used to display SPC results that have nonconforming counts for each sample. It does not retrieve SPC results from the SPC module so it must be used with either the SPC Selector or the SPC Controller components that do. Only SPC results with p chart SPC Data Format specified will be displayed.

Properties

Name	Scripting	Category	Property Type	Description
No Data Font	noDataFont	Appearance	Font	The font to use for the no data message.
No Data Foreground	noDataForeground	Appearance	Color	The foreground color of the no data message.
No Data Message	noDataMessage	Appearance	String	The message to show when there is no data.
Edit Control Limit Image	editControlLimitImagePath	Chart	String	Image to show for editing control limits.
Enable Control Limit Editing	enableControlLimitEditing	Chart	boolean	If true, allow users to change control limit values.
Enable Note Editing	enableNoteEditing	Chart	boolean	If true, allow users to add and edit notes.

Name	Scripting	Category	Property Type	Description
Enable Point Deletion	enablePointDeletion	Chart	boolean	If true, allow users to delete points.
Horizontal Grid Line Color	horzGridLineColor	Chart	Color	The color of horizontal grid lines.
Limit Dialog Horizontal Offset	limitDialogOffsetX	Chart	String	The horizontal offset to display the control limit dialog box.
Limit Dialog Vertical Offset	limitDialogOffsetY	Chart	String	The vertical offset to display the control limit dialog box.
Marker Image Path	markerImagePath	Chart	String	The relative path of an image to display for markers.
Marker Label Font	markerLabelFont	Chart	Font	The font to use for the markers.
Note Image	noteImagePath	Chart	String	Image to show on the the charts when notes exist for a sample.
	primaryChartBackground	Chart	Color	

Name	Scripting	Category	Property Type	Description
Primary Chart Background				The background color of the primary chart.
Right Axis Width	rightAxisWidth	Chart	int	The width of the right chart axis.
Secondary Chart Background	secondaryChartBackground	Chart	Color	The background color of the secondary chart.
Show Horizontal Grid Lines	showHorzGridLines	Chart	boolean	If true, show horizontal grid lines on charts.
Show Notes	showNotes	Chart	boolean	If true, show notes on the chart.
Show Primary Average	showPrimaryAverage	Chart	boolean	Set to true to display the average line on the primary chart.
Show Primary Chart	showPrimaryChart	Chart	boolean	Set to true to display the primary chart.
Show Secondary Average	showSecondaryAverage	Chart	boolean	Set to true to display the average line

Name	Scripting	Category	Property Type	Description
				on the secondary chart.
Show Seconday Chart	showSecondaryChart	Chart	boolean	Set to true to display the secondary chart.
Show Vertical Grid Lines	showVertGridLines	Chart	boolean	If true, show vertical grid lines on charts.
Vertical Grid Line Color	vertGridLineColor	Chart	Color	The color of vertical grid lines.
Calculated Values	calcValues	Data	String	Calculated value definitions.
Measurement Count	measurementCount	Data	int	Number of measurements in the SPC results.
SPC Data	sPCData	Data	String	SPC Data.
SPC Results	sPCResults	Data	String	SPC results containing data, measurement count and calculated value information.

Name	Scripting	Category	Property Type	Description
User	user	Data	String	Current user. If this property is not set, then the currently logged in user will be used.
Calc Background	calcBackground	Table	Color	The background color of the calculated values.
CalcFont	calcFont	Table	Font	The font to use for the calculated values.
Calc Foreground	calcForeground	Table	Color	The foreground color of the calculated values.
Column Width	columnWidth	Table	int	The width of the table columns.
Data Background	dataBackground	Table	Color	The background color of the data values.
Data Font	dataFont	Table	Font	The font to use for the data values.

Name	Scripting	Category	Property Type	Description
Data Foreground	dataForeground	Table	Color	The foreground color of the data values.
Date Background	dateBackground	Table	Color	The background color of the date row.
Date Font	dateFont	Table	Font	The font to use for the date row.
Date Foreground	dateForeground	Table	Color	The foreground color of the date row.
Date Format	dateFormat	Table	String	The date formatting pattern used to display the date.
Enable Highlights	enableHighlights	Table	String	Enables highlighting of signals and control limits in the table.
Label Background	labelBackground	Table	Color	The background color of the labels.
Label Font	labelFont	Table	Font	

Name	Scripting	Category	Property Type	Description
				The font to use for the labels.
Label Foreground	labelForeground	Table	Color	The foreground color of the labels.
Min Visible Samples	minVisibleSamples	Table	String	The minimum number of sample to show in the table.
Row Height	rowHeight	Table	int	The height of the table rows.
Scroll X	scrollX	Table	String	The scroll bar x poition.
Show Table	showTable	Table	boolean	Set to true to display the sample values table.
Visible Measurements	visibleMeasurements	Table	String	The number of measurements to show in the table.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

getNoteToolTip

- Description

Called when tool tip content is generated for a note symbol. The default tool tip is passed and can be modified or replaced.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample that the tool tip is being generated for.

defaultToolTip - The default tool tip that can be modified or replaced.

- Return

The defaultToolTip.

- Scope

Client

getSampleToolTip

- Description

Called when tool tip content is generated for a sample. The default tool tip is passed and can be modified or replaced.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample that the tool tip is being generated for.

valueInfo - The SPCCalcValueInfo object that holds information about the value.

defaultToolTip - The default tool tip that can be modified or replaced.

- Return

The defaultToolTip.

- Scope

Client

beforeSampleDelete

- Description

Called before the sample be deleted. If return value is True, then the sample be deleted.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample to be deleted.

- Return

True

- Scope

Client

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.

Property	Description
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	

Property	Description
	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

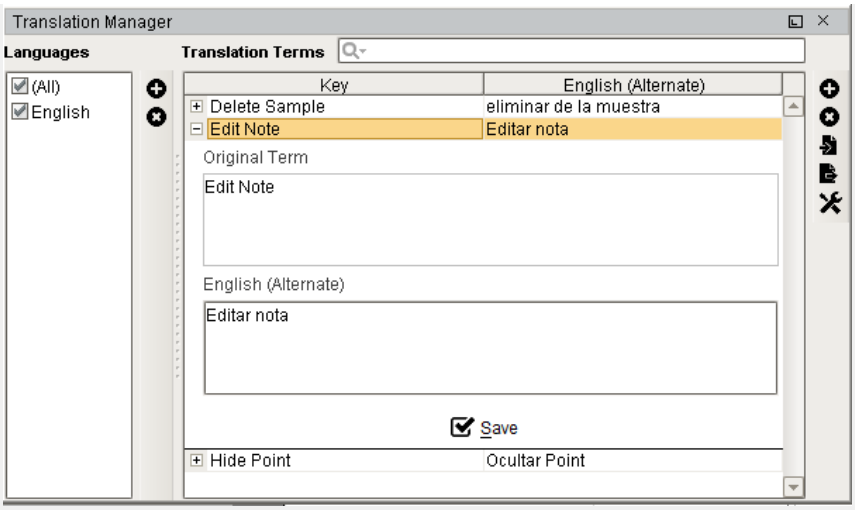
propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Control Chart Menu Items

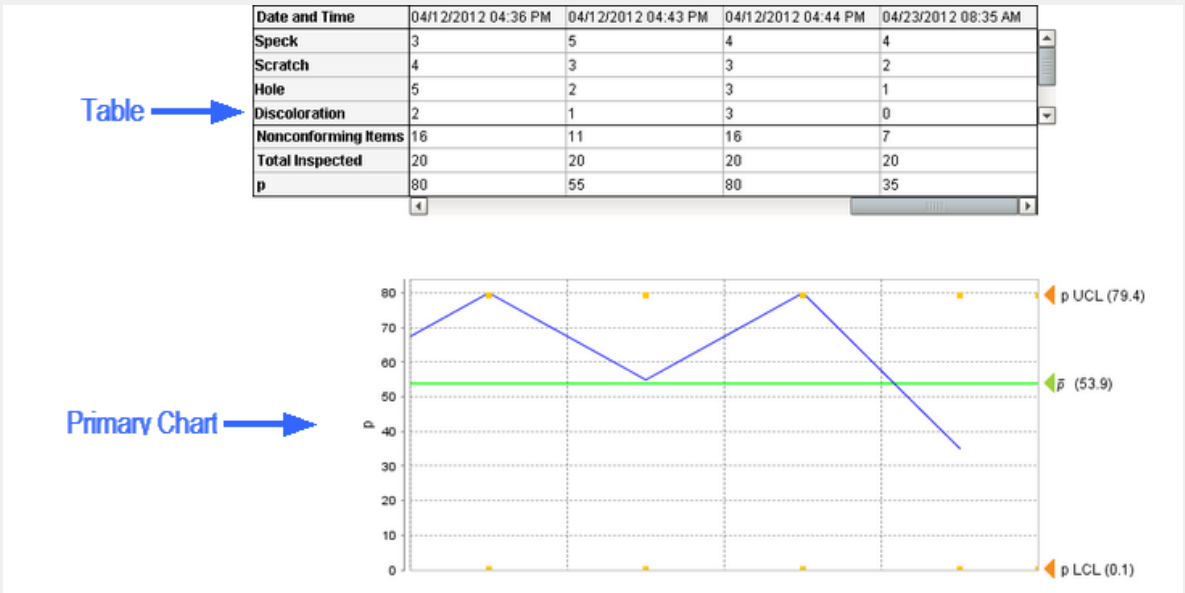
Right clicking the control chart will give the pop up menu items like Delete Sample, Edit Note, Hide Point and Restore Hidden Points. SPC chart popup panel menu strings are localizable. The alternate strings can be added through the Ignition translation manager (not the component translation manager). Reopen the control chart page and right click on a point to manifest the changes.



Customizers

This component does not have any custom properties.

Examples

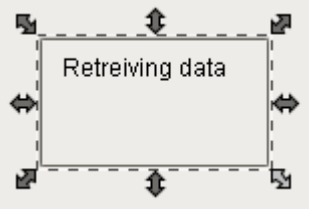


SPCPChart

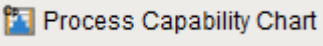
P Control Chart

Process Capability Chart

General



Component Palette Icon:



Description

A component that is used to analyze process capability. The Process Capability is a measurable property of a process to the specification, expressed as a process capability index or as a process performance index.

Properties

Name	Scripting	Category	Property Type	Description
Vertical	vertical	Chart	Boolean	The orientation of the chart.
Chart Background Color	chartBackground	Chart	Color	The color of the chart background.
Bar Color	barColor	Chart	Color	The color of the bars for the chart.
Bar Spacing	barSpacing	Chart	Float8	Spacing between bar as a percentage of the bar width.

Gradient	gradient	Chart	Boolean	If true, bars will be painted with a gradient 'shine'.
Shadows	shadow	Chart	Boolean	If true, show shadows for bars.
Tick Label Font	tickLabelFont	Chart	Font	The font of the tick labels.
Tick Label Color	tickLabelColor	Chart	Color	The color of the tick labels.
Show Bell Curve	showBellCurve	Chart	Boolean	If true, show the bell curve on the chart.
Bell Curve Color	bellCurveColor	Chart	Color	Color of the bell curve line.
Spec Target Color	specTargetColor	Chart	Color	Color of the specification target line.
Spec Limit Color	specLimitColor	Chart	Color	Color of the lsl and usl lines.
Show LCL	showLCL	Chart	Boolean	If true, show the lower control limit on the chart.
Show UCL	showUCL	Chart	Boolean	If true, show the upper control limit on the chart.
Control Limit Color	controlLimitColor	Chart	Color	Color of the lcl and ucl lines.

Name	Scripting	Category	Property Type	Description
Value Axis Title	valueAxisTitle	Chart	String	Title to show for the value axis.
Frequency Axis Title	frequencyAxisTitle	Chart	String	Title to show for the frequency axis.
Axis Title Font	axisTitleFont	Chart	Font	Font for both axis titles.
Axis Title Color	axisTitleColor	Chart	Color	Color to display both axis titles.
Vertical Grid Line Color	vertGridLineColor	Chart	Color	The color of vertical grid lines.
Show Vertical Grid Lines	showVertGridLines	Chart	Boolean	If true, show vertical grid lines on charts.
Horizontal Grid Line Color	horzGridLineColor	Chart	Color	The color of horizontal grid lines.
Show Horizontal Grid Lines	showHorzGridLines	Chart	Boolean	If true, show horizontal grid lines on charts.
Mean	mean	Calculated Values	Float8	The mean value.
Standard Deviation	stdDev	Calculated Values	Float8	The standard deviation value.
lcl	lcl	Calculated Values	Float8	The lower control limit value.
ucl	ucl	Calculated Values	Float8	The upper control limit value.

Name	Scripting	Category	Property Type	Description
cp	cp	Calculated Values	Float8	The cp value.
cpk	cpk	Calculated Values	Float8	The cpk value.
cpm	cpm	Calculated Values	Float8	The cpm value.
cpl	cpl	Calculated Values	Float8	The cpl value.
cpu	cpu	Calculated Values	Float8	The cpu value.

Scripting

Scripting Functions

showSetLimitPanel

- Description

Causes the calculate and set control limit dialog to be shown.

- Parameters

String chartName - Which chart to show the control limit dialog for. Available options are "Primary".

- Return

Nothing

- Scope

Client

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	

Property	Description
	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

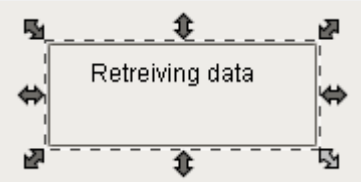
Customizers

This component does not have any custom properties.

Examples

Process Performance Chart

General



Component Palette Icon: Process Performance Chart

Description

A component that is a graph which is used to analyze process performance.

Properties

Name	Scripting	Category	Property Type	Description
Vertical	vertical	Chart	Boolean	The orientation of the chart.
Chart Background Color	chartBackground	Chart	Color	The color of the chart background.
Bar Color	barColor	Chart	Color	The color of the bars for the chart.
Bar Spacing	barSpacing	Chart	Float8	Spacing between bar as a percentage of the bar width.
Gradient	gradient	Chart	Boolean	If true, bars will be painted with a gradient 'shine'.
Shadows	shadow	Chart	Boolean	If true, show shadows for bars.
Tick Label Font	tickLabelFont	Chart	Font	The font of the tick labels.
Tick Label Color	tickLabelColor	Chart	Color	The color of the tick labels.
	showBellCurve	Chart	Boolean	

Name	Scripting	Category	Property Type	Description
Show Bell Curve				If true, show the bell curve on the chart.
Bell Curve Color	bellCurveColor	Chart	Color	Color of the bell curve line.
Spec Target Color	specTargetColor	Chart	Color	Color of the specification target line.
Spec Limit Color	specLimitColor	Chart	Color	Color of the lsl and usl lines.
Show LCL	showLCL	Chart	Boolean	If true, show the lower control limit on the chart.
Show UCL	showUCL	Chart	Boolean	If true, show the upper control limit on the chart.
Control Limit Color	controlLimitColor	Chart	Color	Color of the lcl and ucl lines.
Value Axis Title	valueAxisTitle	Chart	String	Title to show for the value axis.
Frequency Axis Title	frequencyAxisTitle	Chart	String	Title to show for the frequency axis.
Axis Title Font	axisTitleFont	Chart	Font	Font for both axis titles.
Axis Title Color	axisTitleColor	Chart	Color	Color to display both axis titles.
	vertGridLineColor	Chart	Color	

Name	Scripting	Category	Property Type	Description
Vertical Grid Line Color				The color of vertical grid lines.
Show Vertical Grid Lines	showVertGridLines	Chart	Boolean	If true, show vertical grid lines on charts.
Horizontal Grid Line Color	horzGridLineColor	Chart	Color	The color of horizontal grid lines.
Show Horizontal Grid Lines	showHorzGridLines	Chart	Boolean	If true, show horizontal grid lines on charts.
Mean	mean	Calculated Values	Float8	The mean value.
Standard Deviation	stdDev	Calculated Values	Float8	The standard deviation value.
lcl	lcl	Calculated Values	Float8	The lower control limit value.
ucl	ucl	Calculated Values	Float8	The upper control limit value.
pp	pp	Calculated Values	Float8	The pp value.
ppk	ppk	Calculated Values	Float8	The ppk value.
pr	pr	Calculated Values	Float8	The pr value.
ppl	ppl	Calculated Values	Float8	The ppl value.

Name	Scripting	Category	Property Type	Description
ppu	ppu	Calculated Values	Float8	The ppu value.

Scripting

Scripting Functions

showSetLimitPanel

- Description

Causes the calculate and set control limit dialog to be shown.

- Parameters

String chartName - Which chart to show the control limit dialog for. Available options are "Primary".

- Return

Nothing

- Scope

Client

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	**Description**
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

Property	Description
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	

Property	Description
	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

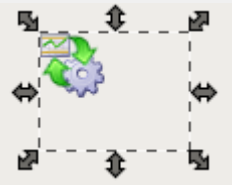
Customizers

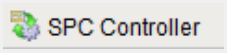
This component does not have any custom properties.

Examples

SPC Controller

General



Component Palette Icon: 

Description

An invisible component that makes SPC data available for reports and other components. The term invisible component means that this component appears during design time, but is not visible during runtime.

Properties

Name	Scripting	Category	Property Type	D
Automatic Update	automaticUpdate	Data	Boolean	If a u p v c
Auto Refresh	autoRefresh	Data	Boolean	If d s d in c
Row Limit	rowLimit	Data	Int4	T n s re re
Stored SPC Name	storedSPCName	Data	String	C u S C th s S
Definition Name	definitionName	Data	String	S D N
Definition Version	definitionVersion	Data	Int4	S D V
Attribute Name	attributeName	Data	String	A N

Name	Scripting	Category	Property Type	D
AttributeUnits	attributeUnits	Data	String	T o s a
AttributeDefaultChart	attributeDefaultChart	Data	String	T c c a
Use Default Chart Type	useDefaultChartType	Data	Boolean	If th c fr a d
Filter	filters	Data	String	F
Control Limits	controlLimits	Data	String	C
Signals	signals	Data	String	S
Additional Factors	additionalFactors	Data	String	A fa in d
Include Disabled Attributes	includeDisabledAttributes	Data	Boolean	If a a w in
Start Date	startDate	Data	DateTime	S
End Date	endDate	Data	DateTime	E

Name	Scripting	Category	Property Type	D
SPC Results	sPCResults	Data	SPCResults	S ir m c c v ir
SPC Data	sPCData	Data	DataSet	S
Measurement Count	measurementCount	Data	int	N m ir re
Calculated Values	calcValues	Data	SPCCalcValueCollection	C v d
Error Message	errorMessage	Data	String	E m
Warning Message	warningMessage	Data	String	W m
Auto Bar Count	autoBarCount	Histogram	Boolean	If n d ir h re a c
Data Bar Count	dataBarCount	Histogram	Int4	If C th d

Name	Scripting	Category	Property Type	D
				ir h re
Padding Bar Count	paddingBarCount	Histogram	Int4	T b a th to th re

Scripting

Scripting Functions

update

- Description

Causes the SPC results to be updated.

- Parameters

None

- Return

Nothing

- Scope

Client

refreshInfo

- Description

Causes the sample definition information to be refreshed.

- Parameters

None

- Return

Nothing

- Scope

Client

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

sPCUpdate

beforeUpdate

Is fired just before SPC results are requested from the SPC module.

Property	Description
source	The component that fired this event.

afterUpdate

Is fired just after SPC results are requested from the SPC module.

Property	Description
source	The component that fired this event.

Customizers

This component does not have any custom properties.

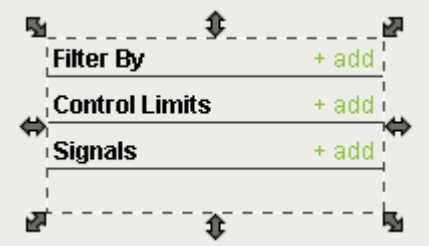
Examples

In cases where the SPC Selector offers too many options to the user, this component can be used. It has all of the same functionality as the SPC Selector but without the user interface. This means property bindings or script must be used to make the filter, compare by and data point selections. It also is used for providing data to canned reports and optionally allowing the user to make limited filter options.

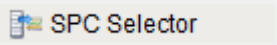
To display the SPC results of this component in a control chart, bind the SPC Results property of the control chart to the SPC Results property of this component.

SPC Selector

General



Component Palette Icon:



Description

A component that allows selections of SPC data. As the user makes selections, this component will query the server for results. These results can be accessed through the SPC Results and SPC Data and can be linked with any of the SPC control charts.

Properties

Name	Scripting	Category	Property Type	Descri
Border	settingsBorder	Common	Border	Border main s panel.

Name	Scripting	Category	Property Type	Descri
Slide Out Background Color	slideOutBackground	Appearance	Color	The backgr color o slide o
Slide Out Border	slideOutBorder	Appearance	Border	Border slide o panel.
Min Slide Out Width	minSlideOutWidth	Appearance	int	The ma width c slide o
Max Slide Out Width	maxSlideOutWidth	Appearance	int	The ma width c slide o
Padding	padding	Appearance	int	The an paddin betwee notes.
Header Background Color	headerBackground	Appearance	Color	The co the hea backgr
Start Date	startDate	Data	DateTime	Start D
End Date	endDate	Data	DateTime	End Da
Auto Refresh	autoRefresh	Data	Boolean	If true, data w sample definiti informa change
SPC Results	sPCResults	Data	SPCResults	

Name	Scripting	Category	Property Type	Descri
				SPC R includi measu count a calcula value inform
SPC Data	sPCData	Data	DataSet	SPC d
Measurement Count	measurementCount	Data	int	Numbe measu in the S results
Calculated Values	calcValues	Data	SPCCalcValueCollection	Calcula value definiti
Message	message	Data	String	Error o warnin messa associ with th results
Suppress Warnings	suppressWarnings	Data	Boolean	If true, error m dialog
Suppress Errors	suppressErrors	Data	Boolean	If true, warnin messa dialog
Include Disabled Attributes	includeDisabledAttributes	Data	Boolean	

Name	Scripting	Category	Property Type	Descri
				If true, attribut are dis will be include
Filter Selection Summary	filterSummary	Data	String	Summ your fil selecti string.
Control Limit Summary	controlLimitSummary	Data	String	Summ your co limit se in a str
Signal Summary	signalSummary	Data	String	Summ your si selecti string.
Attribute Name	attributeName	Data	String	The na the cur sample attribut
Attribute Units	attributeUnits	Data	String	The un of the sample attribut
Attribute Default Chart	attributeDefaultChart	Data	String	The de chart o current attribut
Use Default Chart Type	useDefaultChartType	Data	Boolean	

Name	Scripting	Category	Property Type	Descri
				If true, the def chart ty from th attribut definiti
Definition Name	definitionName	Data	String	The cu definiti name.
Auto Bar Count	autoBarCount	Histogram	boolean	If true, numbe data ba include histogr results automa calcula
Data Bar Count	dataBarCount	Histogram	Int4	If Auto Count the nur data ba include histogr results
Padding Bar Count	paddingBarCount	Histogram	Int4	The nu bars pr and fol the dat to inclu the his results

Scripting

Scripting Functions

refreshInfo

- Description

Force refresh of the SPC results.

- Parameters

None

- Return

Nothing

- Scope

Client

setSpcDataFormat

- Description

Change to format if the SPC data to return.

- Parameters

spcDataFormat - Format of the SPC data to return.

Data Type	**int**
None	0
XBarR	1
XBarS	2
Individual	3
Median	4
U	5
C	6

P	7
NP	8
Histogram	9
Pareto	10

- Return

Nothing

- Scope

Client

setRowLimit

- Description

Change the default number of samples to return to the value specified in the rowLimit parameter. By default only 500 samples are returned in the SPC results. This is done to unburden the database, network bandwidth and memory.

- Parameters

int rowLimit - New row limit.

- Return

Nothing

- Scope

Client

getRowLimit

- Description

Returns the current row limit value.

- Parameters

None

- Return

Nothing

- Scope

Client

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	

Property	Description
	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

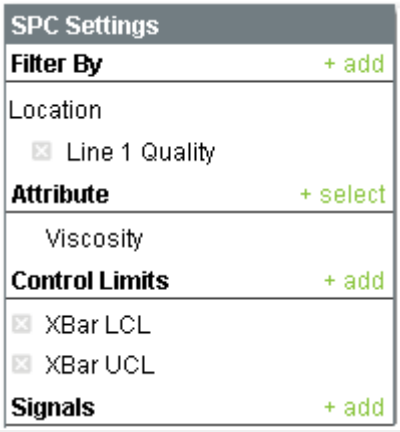
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

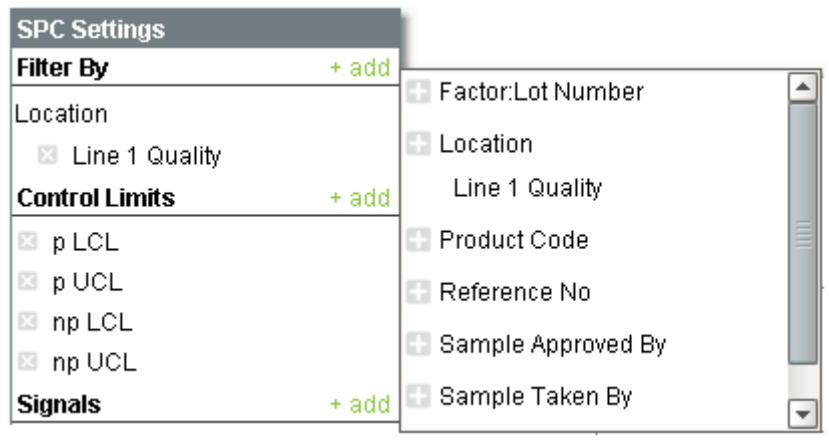
This component does not have any custom properties.

Examples



SPC Selector

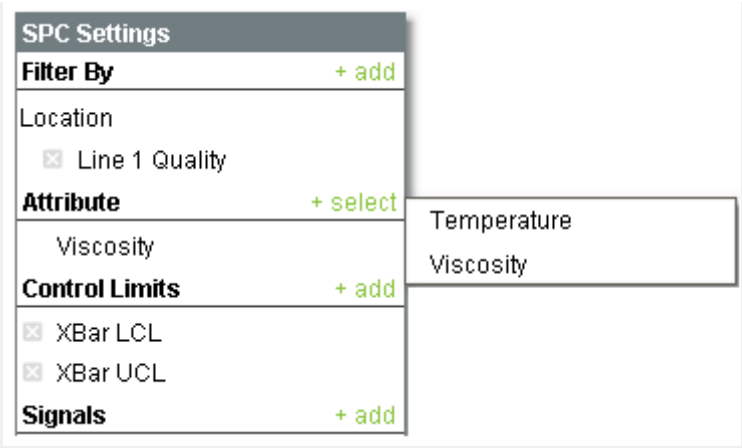
A filter can be added by selecting the + add link to the right of **Filter By**. A window panel will open and filter categories will be displayed. Click the link by the filter category and specific filter items will be displayed. When selected they will be added to the filters as shown below. To minimize the number of filter options, reduce the date range defined by the Start Date and End Date properties and the associated filter values will be shown. Because values collected from different locations being shown together does not make sense, a location must be added to the Filter By section.



Filter By List

Sample definitions can have more than one attribute. At the time sample data is recorded, each attribute will have a value associated with it. For example, when collecting viscosity reading it may also be important to know the temperature. But, showing and making calculations on a viscosity value of 10000 with a temperature value of 75.2 does not make sense. The SPC Selector allow selecting a single attribute as shown below.

If an attribute type of sample definition is selected, then the Attribute section will not appear. This is because with attribute charts, all attributes are included as shown. For example, if a sample definition has an attribute for Torn, Discolored, Pitted, etc. then all will show in the table and is included for calculations.



Attribute Selection

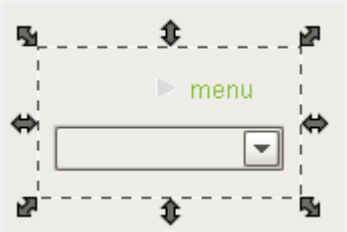
Similar to filters, control limits and signals can be added to the SPC results. Any selected control limits, and signals that depend on them, will not appear on the control chart until the control limit value has been set.

Selections can be removed by selecting the link to the left of the selection.

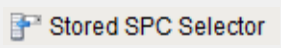
To display the SPC results of this component in a control charts, bind the SPC Results property of the control chart to the SPC Results property of this component.

Stored SPC Selector

General



Component Palette Icon:



Description

A component that allows creating, recalling and saving SPC selections in the SPC Selector component. This component will automatically use the available SPC Selector in the container. Keep in mind that whenever a new sample definition is created, the new stored SPC settings items will be created with the default values. This being said, additional stored SPC settings items can be created each with different filters, attribute, control limits and signals.

Properties

Name	Scripting	Category	Property Type	Description
Show Disabled	showDisabled	Appearance	Boolean	If true, include disabled definitions.
Show Menu	showMenu	Appearance	Boolean	Display the menu for the selector.
Menu Top Position	menuTopPos	Appearance	Int4	The top position of the menu.
Menu Left Position	menuLeftPos	Appearance	Int4	The left position of the menu.
Menu Image	menuImagePath	Appearance	String	Image to show for the menu.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

Event Handlers

menu

userMenuItemClicked

This event fires when the menu item is clicked, or if the user selects the menu item using the keyboard and presses the Enter key. It can also occur if an access key or shortcut key is pressed that is associated with the MenuItem.

Property	Description
source	The component that fired this event.
menuItemName	Name of the user menu item that triggered the event.
nodeName	Name of the node. This is the same as the name of the MES object that is associated with the node.
objectType	Name of the MES object type that is associated with the node.
uuid	UUID of the MES object that is associated to the node.
lotUUID	UUID of the material lot.
lotName	Name of the material lot.
lotSequence	The sequence number associated with the material lot.
lotUse	The lot use type of the material.

Property	Description
beginDateTime	Date and Time at which the event was triggered.
materialUUID	UUID of the material.
materialName	Name of the material.
lotEquipmentUUID	UUID of the equipment lot.
lotEquipmentName	Name of the equipment lot.
segmentUUID	UUID of the segment.
segmentName	Name of the segment.
segmentEquipmentUUID	UUID of the segment equipment.
segmentEquipmentName	Name of the segment equipment.

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	

Property	Description
	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.

Property	Description
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	

Property	Description
	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	

Property	Description
	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

settings

selected

Is fired when a different SPC Settings item is selected menu item is selected.

Property	Description
source	The component that fired this event.
settingsName	The name of the newly selected SPC Settings item.
settings	The SPCSettings object that contains the filter, attribute, control limit and signal selections.
prevName	The previous name of the SPC Settings item.

created

Property	Description
source	The component that fired this event.
settingsName	The name of the newly selected SPC Settings item.
settings	The SPCSettings object that contains the filter, attribute, control limit and signal selections.
prevName	The previous name of the SPC Settings item.

deleted

Property	Description
source	The component that fired this event.
settingsName	The name of the newly selected SPC Settings item.
settings	The SPCSettings object that contains the filter, attribute, control limit and signal selections.
prevName	The previous name of the SPC Settings item.

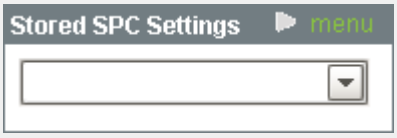
renamed

Property	Description
source	The component that fired this event.
settingsName	The name of the newly selected SPC Settings item.
settings	The SPCSettings object that contains the filter, attribute, control limit and signal selections.
prevName	The previous name of the SPC Settings item.

Customizers

This component does not have any custom properties.

Examples



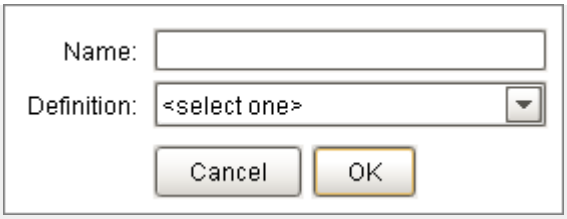
Stored SPC Settings

Stored SPC Selector

By clicking on the

link, a menu with the option to create new, save, delete and rename SPC settings will popup.

To add a new saved SPC settings item, click on **New** menu item, enter a name, select a sample definition and click **OK**. This will create a default SPC Settings item. Now the user can select filters, attribute, control limits and signals that will be saved and can easily be selected at a later time.



StoredSPCSettingsNew

New Stored SPC Settings

To rename a stored SPC Settings item, select an item and click on the **Rename** menu item, enter a new name and click **OK**.

Component Stored Analyis Rename

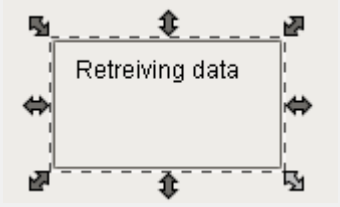
Rename Stored SPC Settings

To delete a stored SPC Settings item, select an item and click on **Delete** menu item, and select **Yes** to the confirmation message.

If changes to a stored SPC settings values have been made and the user selects a different stored SPC Settings, they will be prompted to save the changes. Alternatively, the changes can be saved by clicking on the **Save** menu item.

U-Chart

General



Component Palette Icon: U-Chart

Description

The Percentage of Nonconformities (u) control chart is used to display SPC results that have nonconformities counts for each sample. It does not retrieve SPC results from the SPC module so it must be used with either the SPC Selector or the SPC Controller components that do. Only SPC results with u chart SPC Data Format specified will be displayed.

Properties

Name	Scripting	Category	Property Type	Description
No Data Font	noDataFont	Appearance	Font	The font to use for the no data message.
No Data Foreground	noDataForeground	Appearance	Color	The foreground color of the no data message.
No Data Message	noDataMessage	Appearance	String	The message to show when there is no data.
Edit Control Limit Image	editControlLimitImagePath	Chart	String	Image to show for editing control limits.
Enable Control Limit Editing	enableControlLimitEditing	Chart	boolean	If true, allow users to change control limit values.
Enable Note Editing	enableNoteEditing	Chart	boolean	If true, allow users to add and edit notes.
	enablePointDeletion	Chart	boolean	

Name	Scripting	Category	Property Type	Description
Enable Point Deletion				If true, allow users to delete points.
Horizontal Grid Line Color	horzGridLineColor	Chart	Color	The color of horizontal grid lines.
Limit Dialog Horizontal Offset	limitDialogOffsetX	Chart	String	The horizontal offset to display the control limit dialog box.
Limit Dialog Vertical Offset	limitDialogOffsetY	Chart	String	The vertical offset to display the control limit dialog box.
Marker Image Path	markerImagePath	Chart	String	The relative path of an image to display for markers.
Marker Label Font	markerLabelFont	Chart	Font	The font to use for the markers.
Note Image	noteImagePath	Chart	String	Image to show on the the charts when notes exist for a sample.
	primaryChartBackground	Chart	Color	

Name	Scripting	Category	Property Type	Description
Primary Chart Background				The background color of the primary chart.
Right Axis Width	rightAxisWidth	Chart	int	The width of the right chart axis.
Secondary Chart Background	secondaryChartBackground	Chart	Color	The background color of the secondary chart.
Show Horizontal Grid Lines	showHorzGridLines	Chart	boolean	If true, show horizontal grid lines on charts.
Show Notes	showNotes	Chart	boolean	If true, show notes on the chart.
Show Primary Average	showPrimaryAverage	Chart	boolean	Set to true to display the average line on the primary chart.
Show Primary Chart	showPrimaryChart	Chart	boolean	Set to true to display the primary chart.
Show Secondary Average	showSecondaryAverage	Chart	boolean	Set to true to display the average line

Name	Scripting	Category	Property Type	Description
				on the secondary chart.
Show Seconday Chart	showSecondaryChart	Chart	boolean	Set to true to display the secondary chart.
Show Vertical Grid Lines	showVertGridLines	Chart	boolean	If true, show vertical grid lines on charts.
Vertical Grid Line Color	vertGridLineColor	Chart	Color	The color of vertical grid lines.
Calculated Values	calcValues	Data	String	Calculated value definitions.
Measurement Count	measurementCount	Data	int	Number of measurements in the SPC results.
SPC Data	sPCData	Data	String	SPC Data.
SPC Results	sPCResults	Data	String	SPC results containing data, measurement count and calculated value information.

Name	Scripting	Category	Property Type	Description
User	user	Data	String	Current user. If this property is not set, then the currently logged in user will be used.
Calc Background	calcBackground	Table	Color	The background color of the calculated values.
CalcFont	calcFont	Table	Font	The font to use for the calculated values.
Calc Foreground	calcForeground	Table	Color	The foreground color of the calculated values.
Column Width	columnWidth	Table	int	The width of the table columns.
Data Background	dataBackground	Table	Color	The background color of the data values.
Data Font	dataFont	Table	Font	The font to use for the data values.

Name	Scripting	Category	Property Type	Description
Data Foreground	dataForeground	Table	Color	The foreground color of the data values.
Date Background	dateBackground	Table	Color	The background color of the date row.
Date Font	dateFont	Table	Font	The font to use for the date row.
Date Foreground	dateForeground	Table	Color	The foreground color of the date row.
Date Format	dateFormat	Table	String	The date formatting pattern used to display the date.
Enable Highlights	enableHighlights	Table	String	Enables highlighting of signals and control limits in the table.
Label Background	labelBackground	Table	Color	The background color of the labels.
Label Font	labelFont	Table	Font	

Name	Scripting	Category	Property Type	Description
				The font to use for the labels.
Label Foreground	labelForeground	Table	Color	The foreground color of the labels.
Min Visible Samples	minVisibleSamples	Table	String	The minimum number of sample to show in the table.
Row Height	rowHeight	Table	int	The height of the table rows.
Scroll X	scrollX	Table	String	The scroll bar x poition.
Show Table	showTable	Table	boolean	Set to true to display the sample values table.
Visible Measurements	visibleMeasurements	Table	String	The number of measurements to show in the table.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

getNoteToolTip

- Description

Called when tool tip content is generated for a note symbol. The default tool tip is passed and can be modified or replaced.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample that the tool tip is being generated for.

defaultToolTip - The default tool tip that can be modified or replaced.

- Return

The defaultToolTip.

- Scope

getSampleToolTip

- Description

Called when tool tip content is generated for a sample. The default tool tip is passed and can be modified or replaced.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample that the tool tip is being generated for.

valueInfo - The SPCCalcValueInfo object that holds information about the value.

defaultToolTip - The default tool tip that can be modified or replaced.

- Return

The defaultToolTip.

- Scope

Client

beforeSampleDelete

- Description

Called before the sample be deleted. If return value is True, then the sample be deleted.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample to be deleted.

- Return

True

- Scope

Client

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.

Property	Description
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	

Property	Description
	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

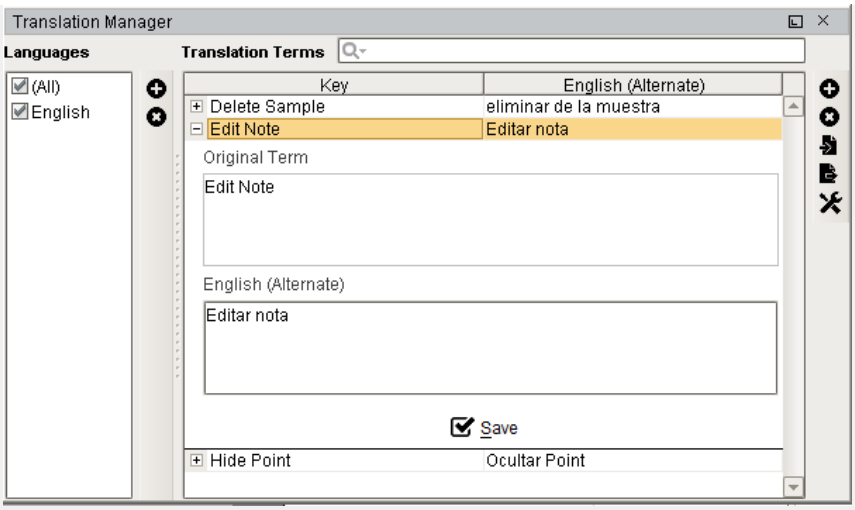
propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Control Chart Menu Items

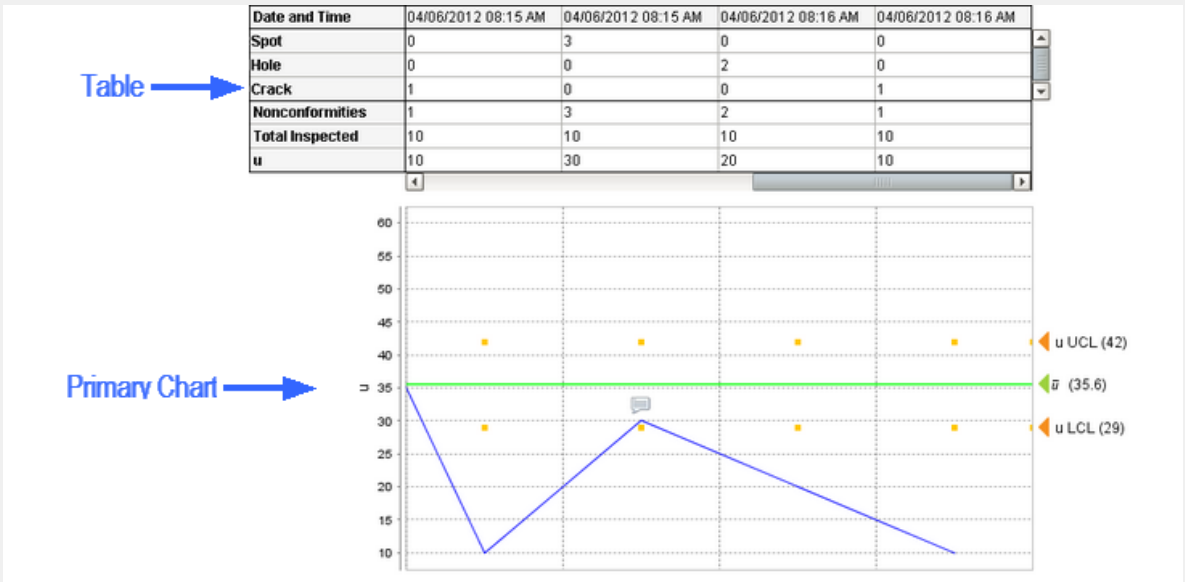
Right clicking the control chart will give the pop up menu items like Delete Sample, Edit Note, Hide Point and Restore Hidden Points. SPC chart popup panel menu strings are localizable. The alternate strings can be added through the Ignition translation manager (not the component translation manager). Reopen the control chart page and right click on a point to manifest the changes.



Customizers

This component does not have any custom properties.

Examples

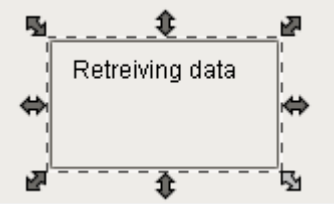


SPCUChart

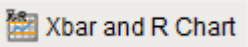
U Control Chart

Xbar and R Chart

General



Component Palette Icon:



Description

The XBar Range control chart is used to display SPC results that have multiple measurements for each sample. It does not retrieve SPC results from the SPC module so it must be used with either the SPC Selector or the SPC Controller components that do. Only SPC results with XBar and Range SPC Data Format specified will be displayed.

Properties

Name	Scripting	Category	Property Type	Description
No Data Font	noDataFont	Appearance	Font	The font to use for the no data message.
No Data Foreground	noDataForeground	Appearance	Color	The foreground color of the no data message.
No Data Message	noDataMessage	Appearance	String	

Name	Scripting	Category	Property Type	Description
				The message to show when there is no data.
Edit Control Limit Image	editControlLimitImagePath	Chart	String	Image to show for editing control limits.
Enable Control Limit Editing	enableControlLimitEditing	Chart	boolean	If true, allow users to change control limit values.
Enable Note Editing	enableNoteEditing	Chart	boolean	If true, allow users to add and edit notes.
Enable Point Deletion	enablePointDeletion	Chart	boolean	If true, allow users to delete points.
Horizontal Grid Line Color	horzGridLineColor	Chart	Color	The color of horizontal grid lines.
Limit Dialog Horizontal Offset	limitDialogOffsetX	Chart	String	The horizontal offset to display the control limit dialog box.
Limit Dialog Vertical Offset	limitDialogOffsetY	Chart	String	The vertical offset to display the control limit dialog box.

Name	Scripting	Category	Property Type	Description
Marker Image Path	markerImagePath	Chart	String	The relative path of an image to display for markers.
Marker Label Font	markerLabelFont	Chart	String	The font to use for the markers.
Note Image	noteImagePath	Chart	String	Image to show on the the charts when notes exist for a sample.
Primary Chart Background	primaryChartBackground	Chart	Color	The background color of the primary chart.
Right Axis Width	rightAxisWidth	Chart	int	The width of the right chart axis.
Secondary Chart Background	secondaryChartBackground	Chart	Color	The background color of the secondary chart.
Show Horizontal Grid Lines	showHorzGridLines	Chart	boolean	If true, show horizontal grid lines on charts.
Show Notes	showNotes	Chart	boolean	

Name	Scripting	Category	Property Type	Description
				If true, show notes on the chart.
Show Primary Average	showPrimaryAverage	Chart	boolean	Set to true to display the average line on the primary chart.
Show Primary Chart	showPrimaryChart	Chart	boolean	Set to true to display the primary chart.
Show Secondary Average	showSecondaryAverage	Chart	boolean	Set to true to display the average line on the secondary chart.
Show Seconday Chart	showSecondaryChart	Chart	boolean	Set to true to display the secondary chart.
Show Vertical Grid Lines	showVertGridLines	Chart	boolean	If true, show vertical grid lines on charts.
Vertical Grid Line Color	vertGridLineColor	Chart	Color	The color of vertical grid lines.
Calculated Values	calcValues	Data	String	

Name	Scripting	Category	Property Type	Description
				Calculated value definitions.
Measurement Count	measurementCount	Data	String	Number of measurements in the SPC results.
SPC Data	sPCData	Data	String	SPC Data.
SPC Results	sPCResults	Data	String	SPC results containing data, measurement count and calculated value information.
User	user	Data	String	Current user. If this property is not set, then the currently logged in user will be used.
Calc Background	calcBackground	Table	Color	The background color of the calculated values.
CalcFont	calcFont	Table	Font	The font to use for the calculated values.

Name	Scripting	Category	Property Type	Description
Calc Foreground	calcForeground	Table	Color	The foreground color of the calculated values.
Column Width	columnWidth	Table	int	The width of the table columns.
Data Background	dataBackground	Table	Color	The background color of the data values.
Data Font	dataFont	Table	Font	The font to use for the data values.
Data Foreground	dataForeground	Table	Color	The foreground color of the data values.
Date Background	dateBackground	Table	Color	The background color of the date row.
Date Font	dateFont	Table	Font	The font to use for the date row.
Date Foreground	dateForeground	Table	Color	The foreground color of the date row.

Name	Scripting	Category	Property Type	Description
Date Format	dateFormat	Table	String	The date formatting pattern used to display the date.
Enable Highlights	enableHighlights	Table	String	Enables highlighting of signals and control limits in the table.
Label Background	labelBackground	Table	Color	The background color of the labels.
Label Font	labelFont	Table	Font	The font to use for the labels.
Label Foreground	labelForeground	Table	Color	The foreground color of the labels.
Min Visible Samples	minVisibleSamples	Table	String	The minimum number of sample to show in the table.
Row Height	rowHeight	Table	int	The height of the table rows.
Scroll X	scrollX	Table	String	The scroll bar x poition.

Name	Scripting	Category	Property Type	Description
Show Table	showTable	Table	boolean	Set to true to display the sample values table.
Visible Measurements	visibleMeasurements	Table	String	The number of measurements to show in the table.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

getNoteToolTip

- Description

Called when tool tip content is generated for a note symbol. The default tool tip is passed and can be modified or replaced.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample that the tool tip is being generated for.

defaultToolTip - The default tool tip that can be modified or replaced.

- Return

The defaultToolTip.

- Scope

Client

getSampleToolTip

- Description

Called when tool tip content is generated for a sample. The default tool tip is passed and can be modified or replaced.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample that the tool tip is being generated for.

valueInfo - The SPCCalcValueInfo object that holds information about the value.

defaultToolTip - The default tool tip that can be modified or replaced.

- Return

The defaultToolTip.

- Scope

Client

beforeSampleDelete

- Description

Called before the sample be deleted. If return value is True, then the sample be deleted.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample to be deleted.

- Return

True

- Scope

Client

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	**Description**
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

Property	Description
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

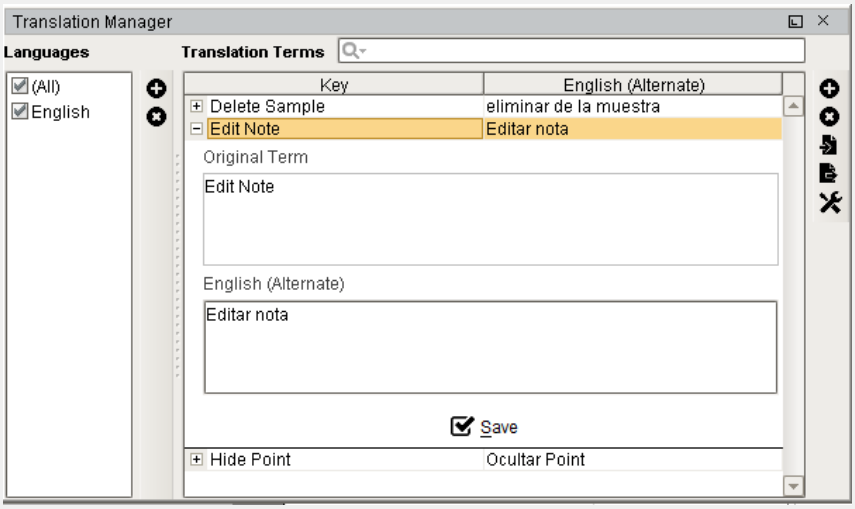
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	

Property	Description
	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Control Chart Menu Items

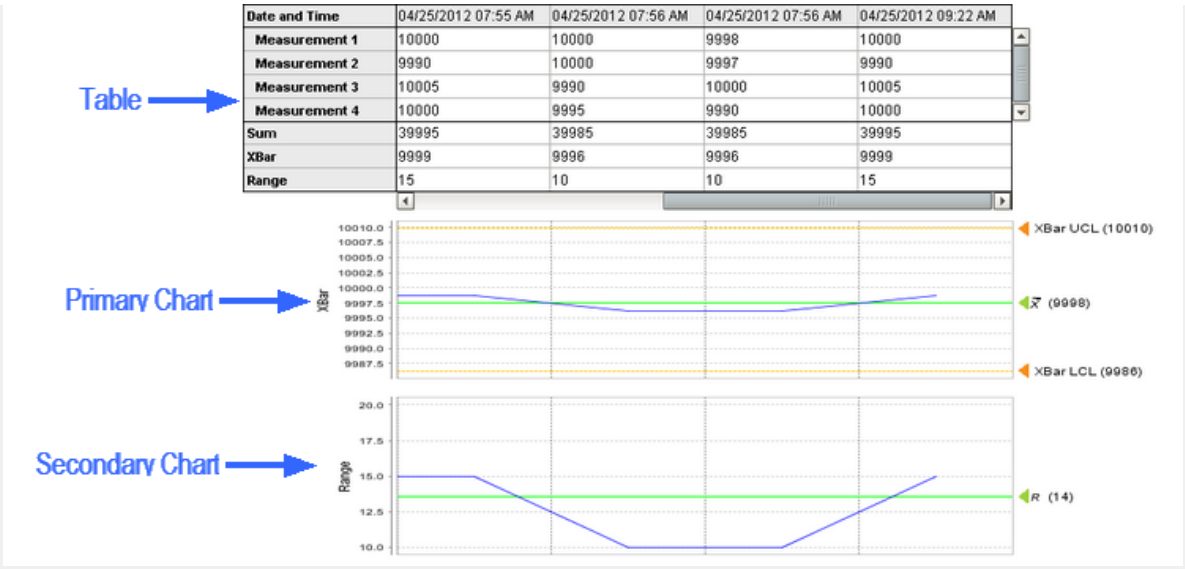
Right clicking the control chart will give the pop up menu items like Delete Sample, Edit Note, Hide Point and Restore Hidden Points. SPC chart popup panel menu strings are localizable. The alternate strings can be added through the Ignition translation manager (not the component translation manager). Reopen the control chart page and right click on a point to manifest the changes.



Customizers

This component does not have any custom properties.

Examples

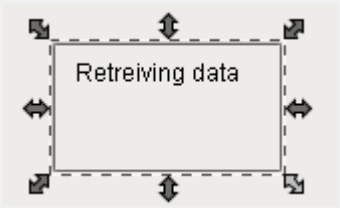


SPCXBarR

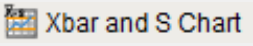
XBar Range Control Chart

Xbar and S Chart

General



Component Palette Icon:



Description

The XBar Standard Deviation (S) control chart is used to display SPC results that have multiple measurements for each sample. It does not retrieve SPC results from the SPC module so it must be used with either the SPC Selector or the SPC Controller components that do. Only SPC results with XBar and S SPC Data Format specified will be displayed.

Properties

Name	Scripting	Category	Property Type	Description
No Data Font	noDataFont	Appearance	Font	The font to use for the no data message.
No Data Foreground	noDataForeground	Appearance	Color	The foreground color of the no data message.
No Data Message	noDataMessage	Appearance	String	The message to show when there is no data.
Edit Control Limit Image	editControlLimitImagePath	Chart	String	Image to show for editing control limits.
Enable Control Limit Editing	enableControlLimitEditing	Chart	boolean	If true, allow users to change control limit values.
Enable Note Editing	enableNoteEditing	Chart	boolean	If true, allow users to add and edit notes.
Enable Point Deletion	enablePointDeletion	Chart	boolean	If true, allow users to delete points.
Horizontal Grid Line Color	horzGridLineColor	Chart	Color	The color of horizontal grid lines.

Name	Scripting	Category	Property Type	Description
Limit Dialog Horizontal Offset	limitDialogOffsetX	Chart	String	The horizontal offset to display the control limit dialog box.
Limit Dialog Vertical Offset	limitDialogOffsetY	Chart	String	The vertical offset to display the control limit dialog box.
Marker Image Path	markerImagePath	Chart	String	The relative path of an image to display for markers.
Marker Label Font	markerLabelFont	Chart	Font	The font to use for the markers.
Note Image	noteImagePath	Chart	String	Image to show on the the charts when notes exist for a sample.
Primary Chart Background	primaryChartBackground	Chart	Color	The background color of the primary chart.
Right Axis Width	rightAxisWidth	Chart	int	The width of the right chart axis.

Name	Scripting	Category	Property Type	Description
Secondary Chart Background	secondaryChartBackground	Chart	Color	The background color of the secondary chart.
Show Horizontal Grid Lines	showHorzGridLines	Chart	boolean	If true, show horizontal grid lines on charts.
Show Notes	showNotes	Chart	boolean	If true, show notes on the chart.
Show Primary Average	showPrimaryAverage	Chart	boolean	Set to true to display the average line on the primary chart.
Show Primary Chart	showPrimaryChart	Chart	boolean	Set to true to display the primary chart.
Show Secondary Average	showSecondaryAverage	Chart	boolean	Set to true to display the average line on the secondary chart.
Show Seconday Chart	showSecondaryChart	Chart	boolean	Set to true to display the secondary chart.

Name	Scripting	Category	Property Type	Description
Show Vertical Grid Lines	showVertGridLines	Chart	boolean	If true, show vertical grid lines on charts.
Vertical Grid Line Color	vertGridLineColor	Chart	Color	The color of vertical grid lines.
Calculated Values	calcValues	Data	String	Calculated value definitions.
Measurement Count	measurementCount	Data	int	Number of measurements in the SPC results.
SPC Data	sPCData	Data	String	SPC Data
SPC Results	sPCResults	Data	String	SPC results containing data, measurement count and calculated value information.
User	user	Data	String	Current user. If this property is not set, then the currently logged in user will be used.
	calcBackground	Table	Color	

Name	Scripting	Category	Property Type	Description
Calc Background				The background color of the calculated values.
CalcFont	calcFont	Table	Font	The font to use for the calculated values.
Calc Foreground	calcForeground	Table	Color	The foreground color of the calculated values.
Column Width	columnWidth	Table	int	The width of the table columns.
Data Background	dataBackground	Table	Color	The background color of the data values.
Data Font	dataFont	Table	Font	The font to use for the data values.
Data Foreground	dataForeground	Table	Color	The foreground color of the data values.
Date Background	dateBackground	Table	Color	

Name	Scripting	Category	Property Type	Description
				The background color of the date row.
Date Font	dateFont	Table	Font	The font to use for the date row.
Date Foreground	dateForeground	Table	Color	The foreground color of the date row.
Date Format	dateFormat	Table	String	The date formatting pattern used to display the date.
Enable Highlights	enableHighlights	Table	String	Enables highlighting of signals and control limits in the table.
Label Background	labelBackground	Table	Color	The background color of the labels.
Label Font	labelFont	Table	Font	The font to use for the labels.
Label Foreground	labelForeground	Table	Color	

Name	Scripting	Category	Property Type	Description
				The foreground color of the labels.
Min Visible Samples	minVisibleSamples	Table	String	The minimum number of sample to show in the table.
Row Height	rowHeight	Table	int	The height of the table rows.
Scroll X	scrollX	Table	String	The scroll bar x poition.
Show Table	showTable	Table	boolean	Set to true to display the sample values table.
Visible Measurements	visibleMeasurements	Table	String	The number of measurements to show in the table.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

getNoteToolTip

- Description

Called when tool tip content is generated for a note symbol. The default tool tip is passed and can be modified or replaced.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample that the tool tip is being generated for.

defaultToolTip - The default tool tip that can be modified or replaced.

- Return

The default tool tip.

- Scope

Client

getSampleToolTip

- Description

Called when tool tip content is generated for a sample. The default tool tip is passed and can be modified or replaced.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample that the tool tip is being generated for.

valueInfo - The SPCCalcValueInfo object that holds information about the value.

defaultToolTip - The default tool tip that can be modified or replaced.

- Return

The default tool tip.

- Scope

Client

beforeSampleDelete

- Description

Called before the sample be deleted. If return value is True, then the sample be deleted.

- Parameters

self - A reference to the component that is invoking this function.

sampleUUID - The UUID of the sample to be deleted.

- Return

True

- Scope

Client

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

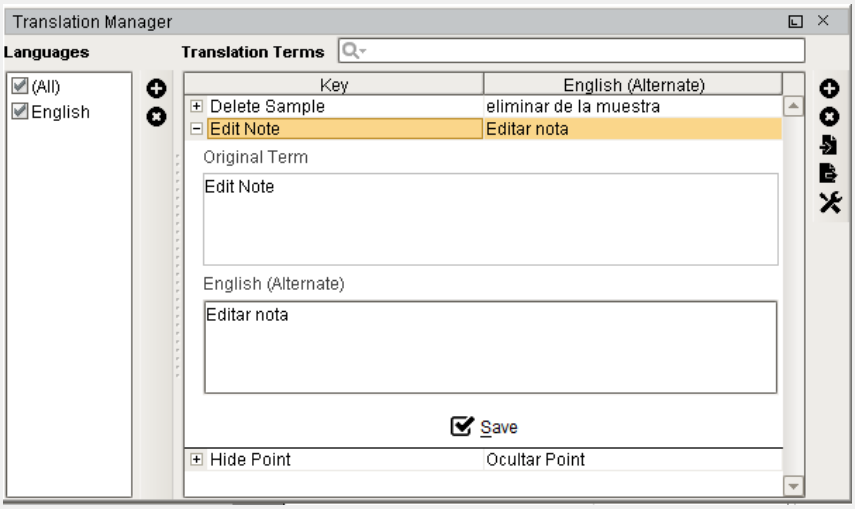
propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Control Chart Menu Items

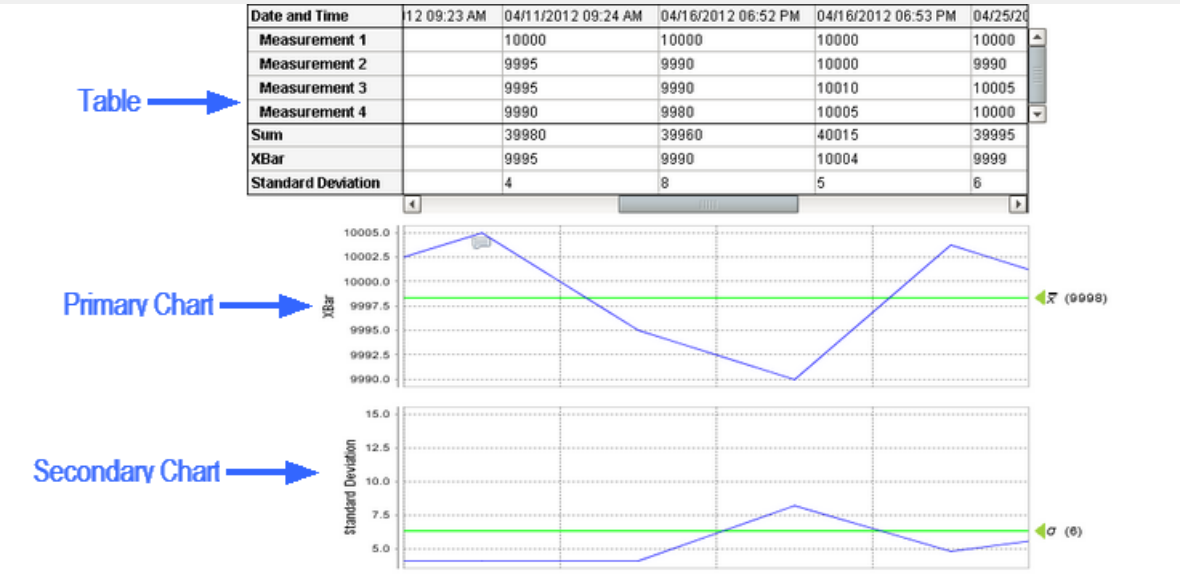
Right clicking the control chart will give the pop up menu items like Delete Sample, Edit Note, Hide Point and Restore Hidden Points. SPC chart popup panel menu strings are localizable. The alternate strings can be added through the Ignition translation manager (not the component translation manager). Reopen the control chart page and right click on a point to manifest the changes.



Customizers

This component does not have any custom properties.

Examples



SPCXBarS

XBar Standard Deviation Control Chart

9.5.5 Recipe Components

Multiple Recipe Editor Table

General

Component Palette Icon:

Description

A component that is added to manage a multiple recipes. This is just one method of managing recipes and for more information on the other methods see the Editing Recipes section.

Properties

Name	Scripting	Category	Property Type	Description
Require Note	requireNote	Behavior	boolean	If true, a user must enter a note when any receipe values are changed.
Read Only	readOnly	Behavior	boolean	If true, a user cannot edit recipe values.
Item Path	itemPath	Data	String	The item path of recipes to view.
Recipe Name Filter	recipeNameFilter	Data	String	Recipe name filter, including * and ? wildcard characters, to filter results by recipe name.
Recipe State Filter	recipeStateFilter	Data	String	

Name	Scripting	Category	Property Type	Description
				Recipe state filter, including * and ? wildcard characters, to filter results by recipe state.
Recipe Group Filter	recipeGroupFilter	Data	String	Recipe group filter, including * and ? wildcard characters, to filter results by recipe group.
Recipe Value Name Filter	recipeValueNameFilter	Data	String	Recipe value name filter, including * and ? wildcard characters, to filter results by recipe value name.
Show Master Recipes	showMasterRecipes	Data	boolean	If true, show master recipes.
Auto-Resize Mode	autoResizeMode	Behavior	boolean	Determines how the table resizes the columns
Selection Foreground	selectionForeground	Appearance	Color	The foreground color of selected cells.
Selection Background	selectionBackground	Appearance	Color	

Name	Scripting	Category	Property Type	Description
				The background color of selected cells.
Grid Line Color	gridColor	Appearance	Color	The grid line color of the table.
Show Horizontal Grid Lines	showHorizontalLines	Appearance	boolean	If true, show horizontal grid lines.
Show Vertical Grid Lines	showVerticalLines	Appearance	boolean	If true, show vertical grid lines.
Show Table Header	headerVisible	Appearance	boolean	If true, show the table header.
Row Height	rowHeight	Appearance	int	The row height in the table.
String Value Display Icon Path	stringValueDisplayIconPath	Appearance	String	The relative path of an icon image appearing for string value display.
String Value Edit Icon Path	stringValueEditIconPath	Appearance	String	The relative path of an icon image appearing for string value edit.
Slide Font	slideFont	Appearance	Font	The font of the slide.
	slideForeground	Appearance	Color	

Name	Scripting	Category	Property Type	Description
Slide Foreground Color				The foreground color of the slide.
Slide Background Color	slideBackground	Appearance	Color	The background color of the slide.
Slide Type	slideType	Appearance	int	The slide type.
Slide Direction	slideDirection	Appearance	int	The slide direction.
Maximum Slide Position	maximumSlidePosition	Appearance	float	The maximum position to open the slide.
Minimum Slide Position	minimumSlidePosition	Appearance	float	The miniimum position to open the slide.
Initial Slide Position	initialSlidePosition	Appearance	float	The initial position to open the slide.
Show Slide Gripper	showSlideGripper	Appearance	boolean	If true, show the slide gripper.

Scripting

Scripting Functions

This component does not have script functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.

Property	Description
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

This component does not have any custom properties.

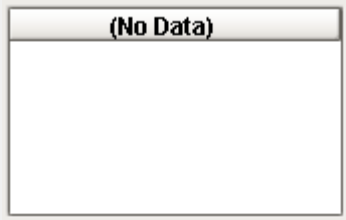
Examples

Value Name	Mixed Nuts 16oz	Mixed Nuts 8oz
MaxWeight	16	8
MinWeight	15	7
TargetWeight	16	8

Property Name	Value
Item Path	[global]\Nuts Unlimited\Folsom\Packaging\Packaging Line 1\Checkweigher

Recipe Change Log Viewer

General



Component Palette Icon: Recipe Changelog Viewer

Description

A component that is added to display recipe change log history in a table. This is just one method of viewing a recipe change log history and for more information on the other methods see the Recipe Change Log section. This component simplifies displaying a recipe changes log by handling all of the backend database queries based on the property settings of the component.

Properties

Name	Scripting	Category	Property Type	Description
Item Path Filter	itemPathFilter	Data	String	Optionally, limit changelog results to production item path.
Recipe Name Filter	recipeNameFilter	Data	String	Optionally, limit changelog results to recipe name.
	recipeValueNameFilter	Data	String	

Name	Scripting	Category	Property Type	Description
Value NameFilter				Optionally, limit changelog results to recipe value name.
User Filter	userFilter	Data	String	Optionally, filter the changelog results by user.
Sub Product Code Filter	subProductCodeFilter	Data	String	Optionally, filter the changelog results by sub product code.
Show Recipe Changes	showRecipeChanges	Data	Boolean	If true, include recipe changelog entries.
Show Default Value Changes	showDefaultValueChanges	Data	Boolean	If true, include default value changelog entries.
Start Date	startDate	Data	DateTime	Start Date.
End Date	endDate	Data	DateTime	End Date.
Data	data	Data	DataSet	The data for the table.
Header Font	headerFont	Appearance	Font	Font of the table's header text.
Header Foreground Color	headerForeground	Appearance	Color	The foreground color of the table's header.

Name	Scripting	Category	Property Type	Description
Row Selection Allowed	rowSelectionAllowed	Behavior	boolean	This flag is used in conjunction with the Column Selection Allowed flag to determine whether not whole-rows, whole-columns, or both.
Header Visible	headerVisible	Appearance	boolean	Whether or not the table header is visible.
Resizing Allowed	resizingAllowed	Behavior	boolean	Whether or not the user is allowed to resize table headers or not.
Row Height	rowHeight	Appearance	int	The height of each row, in pixels.
Odd Row Background	oddBackground	Appearance	Color	The color which odd rows will be colored if background mode is 'Alternating'.
Selection Background	selectionBackground	Appearance	Color	The background color of a selected cell.
Selection Foreground	selectionForeground	Appearance	Color	

Name	Scripting	Category	Property Type	Description
				The foreground color of a selected cell.
Show Horizontal Grid Lines?	showHorizontalLines	Appearance	boolean	Displays horizontal gridlines making it easier to read.
Show Vertical Grid Lines?	showVerticalLines	Appearance	boolean	Displays vertical gridlines making it easier to read.
Grid Line Color	gridColor	Appearance	Color	The color used to draw grid lines.
Column Attributes Data	columnAttributesData	Data	DataSet	The dataset describing the column attributes.
Selected Row	selectedRow	Data	int	The index of the first selected row, or -1 if none.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	

Property	Description
	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

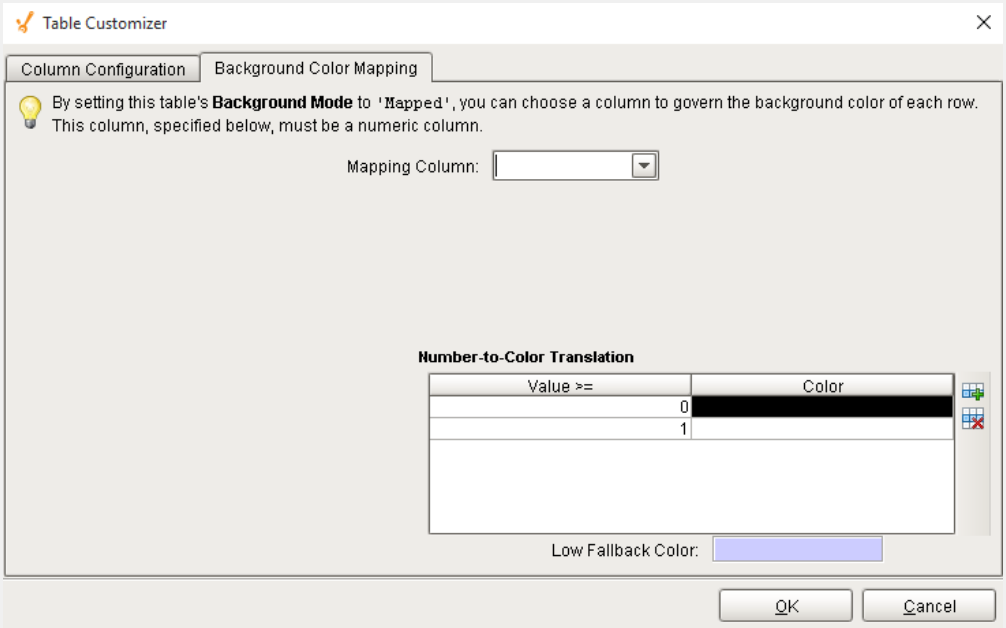
Property	Description
source	The component that fired this event.

Property	Description
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: Remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

Table Customizer

Table Customizer manages the data entered into the Recipe Change Log Viewer.



Examples

Recipe Editor

General

Component Palette Icon:


Recipe Editor

Description

A component that is added to manage recipes. This is just one method of managing recipes and for more information on the other methods see the Editing Recipes section.

Properties

Name	Scripting	Category	Property Type	Description
Item Path Filter	itemPathFilter	Data	String	Optional production item path filter to limit recipe values shown. ? and * can be used as wildcards
Recipe Name Filter	recipeNameFilter	Data	String	Optional recipe name filter to limit recipe values

Name	Scripting	Category	Property Type	Description
				shown. ? and * can be used as wildcards
Recipe Value Name Filter	recipeValueNameFilter	Data	String	Optional recipe value name filter to limit recipe values shown. ? and * can be used as wildcards
Recipe State Filter	recipeStateFilter	Data	String	Optional recipe state name filter to limit recipe values shown. ? and * can be used as wildcards
Recipe Group Filter	recipeGroupFilter	Data	String	Optional recipe group name filter to limit recipe values shown. ? and * can be used as wildcards
Show Item Defaults	showItemDefaults	Data	Boolean	If true, show default values for production items.
	showItemChildren	Data	Boolean	

Name	Scripting	Category	Property Type	Description
Show Item Children				If true, show production item children.
Show Sub Recipes	showSubRecipes	Data	Boolean	If true, show sub recipe values for production items.
Show Recipes	showRecipes	Data	Boolean	If true, show recipes.
Show Master Recipes	showMasterRecipes	Data	Boolean	If true, show master recipes.
Show Descendants	showDescendants	Data	Boolean	If true, show recipe descendants.
Show Recipe Items	showRecipeItems	Data	Boolean	If true, show production items for recipes.
Show Values	showValues	Data	Boolean	If true, show recipe values.
Recipe Value Category	category	Data	String	Category of recipe values to show. Where 1 is recipe values created by the recipe module, 2 is

Name	Scripting	Category	Property Type	Description
				recipe values created by the OEE module and 3 is recipe values created by the SPC module. Use \
Enable Security Editing	enableSecurityEdit	Behavior	Boolean	If true, allow the user to edit recipe value security settings.
Enable Recipe Editing	enableRecipeEdit	Behavior	Boolean	If true, allow the user to add, rename and remove recipes.
Require Note	requireNote	Behavior	Boolean	If true, the user must enter a note when any receipe values are changed.
Read Only	readOnly	Behavior	Boolean	If true, then user cannot select or cancel recipes.
User Menu Items	userMenuItems	Behavior	DataSet	

Name	Scripting	Category	Property Type	Description
				A dataset that stores user menu items.
Default Row Height	defaultRowHeight	Appearance	int	The default row height of the value table.
Max Recipe Value Rows	maxRecipeValueRows	Appearance	int	The maximum number of recipe values to show before scrolling.
Popup Panel Font	panelFont	Appearance	Font	The font to use for the popup panel.
Value Table Font	valueTableFont	Appearance	Font	The font to use for the value table.
Value Table Header Font	valueTableHeaderFont	Appearance	Font	The font to use for the value table header.
Recipes Icon Path	recipesIconPath	Appearance	String	The relative path of an icon image representing multiple recipes.
	recipeIconPath	Appearance	String	

Name	Scripting	Category	Property Type	Description
Recipe Icon Path				The relative path of an icon image representing a single recipe.
Recipe Descendants Icon Path	descendantsIconPath	Appearance	String	The relative path of an icon image representing recipe descendants.
Default Values Icon Path	defaultValuesIconPath	Appearance	String	The relative path of an icon image representing production item default values.
Sub Recipes Icon Path	subRecipesIconPath	Appearance	String	The relative path of an icon image representing multiple sub recipes for a production item.
Sub Recipe Icon Path	subRecipeIconPath	Appearance	String	The relative path of an icon image representing a

Name	Scripting	Category	Property Type	Description
				single sub recipe for a production item.
Default SubRecipe Icon Path	defaultSubRecipeIconPath	Appearance	String	The relative path of an icon image representing the default sub recipe for a production item.
Prod Item Icon Path	prodItemIconPath	Appearance	String	The relative path of an icon image representing a production item.
Menu Add Icon Path	menuAddIconPath	Appearance	String	The relative path of an icon image appearing for the add menu item.
Menu Rename Icon Path	menuRenameIconPath	Appearance	String	The relative path of an icon image appearing for the rename menu item.
Menu Delete Icon Path	menuDeleteIconPath	Appearance	String	

Name	Scripting	Category	Property Type	Description
				The relative path of an icon image appearing for the delete menu item.
Menu Revert Icon Path	menuRevertIconPath	Appearance	String	The relative path of an icon image appearing for the revert menu item.
Menu Security Icon Path	menuSecurityIconPath	Appearance	String	The relative path of an icon image appearing for the security menu item.
Menu Select Items Icon Path	menuSelectItemIconPath	Appearance	String	The relative path of an icon image appearing for the select items menu item.
Note Panel Icon Path	notePanelIconPath	Appearance	String	The relative path of an icon image appearing on the note panel.
	securityPanelIconPath	Appearance	String	

Name	Scripting	Category	Property Type	Description
Security Panel Icon Path				The relative path of an icon image appearing on the security panel.
Item Select Panel Icon Path	itemSelectPanelIconPath	Appearance	String	The relative path of an icon image appearing on the production item select panel.
Note Background Color	noteBackgroundColor	Appearance	Color	The background color of the note panel.
Security Background Color	securityBackgroundColor	Appearance	Color	The background color of the security panel.
Item Selector Background Color	itemSelectorBackgroundColor	Appearance	Color	The background color of the select production item panel.

Scripting

Scripting Functions

changeLocalizationString

- Description

Any of the text that is displayed in the recipe editor change be changed. For example, displaying Recipes for the root recipe node can be replaced with Products. This can be done for any static text in the recipe editor including menu items.

- Parameters

String key - The key to the string value to change.

Recipe Editor component keys:

node.recipes

node.default.values

node.subrecipes

node.subrecipes.default

node.descendants

node.subrecipes.default

menu.recipe.add

menu.recipe.delete

menu.recipe.rename

menu.value.revert

menu.recipe.revertvalues

menu.value.read

menu.recipe.setvalues

menu.value.security

menu.recipe.selectitems

menu.subrecipe.add

menu.subrecipe.delete

menu.subrecipe.rename

menu.recipe.import

menu.recipe.export

panel.item.select.inst

panel.security.inst

panel.note.inst

panel.cancel.desc

panel.ok.desc

String displayText - The new text to replace the existing display text.

- Return

Nothing

- Scope

Client

Code Examples

Code Snippet

```
#Sample script to change the root Recipes node text to
Spanish.

Script from internalFrameActivated event on the window
system.gui.getParentWindow(event).getComponentForPath('Ro
ot Container.Recipe TreeView').changeLocalizationString("
node.recipes", "Receta")
```

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

menu

userMenuItemClicked

This event fires when the menu item is clicked, or if the user selects the menu item using the keyboard and presses the Enter key. It can also occur if an access key or shortcut key is pressed that is associated with the MenuItem.

Property	Description
source	The component that fired this event.
menuItemName	Name of the user menu item that triggered the event.
nodeName	Name of the node. This is the same as the name of the MES object that is associated with the node.
objectType	Name of the MES object type that is associated with the node.
uuid	UUID of the MES object that is associated to the node.
lotUUID	UUID of the material lot.
lotName	Name of the material lot.
lotSequence	The sequence number associated with the material lot.
lotUse	The lot use type of the material.
beginDateTime	Date and Time at which the event was triggered.
materialUUID	UUID of the material.
materialName	Name of the material.
lotEquipmentUUID	UUID of the equipment lot.
lotEquipmentName	Name of the equipment lot.
segmentUUID	UUID of the segment.
segmentName	Name of the segment.

Property	Description
segmentEquipmentUUID	UUID of the segment equipment.
segmentEquipmentName	Name of the segment equipment.

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

Property	Description
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

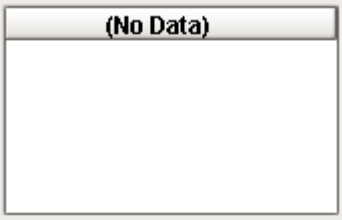
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.

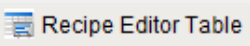
Property	Description
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Recipe Editor Table.

General



Component Palette Icon:



Description

A component that is added to Ignition windows to manage a single recipe. This is just one method of managing recipes and for more information on the other methods see the Editing Recipes section.

Properties

Name	Scripting	Category	Property Type	Description
Item Path	itemPath	Data	String	The item path of the Recipe to view.
Recipe Name	recipeName	Data	String	The name of the Recipe to view.
Recipe Value Name Filter	recipeValueNameFilter	Data	String	The name of the Recipe to view.
Data	data	Data	DataSet	The data for the table.
Live Update Interval	liveValueUpdateInterval	Data	int	The interval in seconds to update the live vale. values less than 1 will not update the live values.
Header Font	headerFont	Appearance	Font	Font of the table's header text.
Header Foreground Color	headerForeground	Appearance	Color	The foreground color of the table's header.
Row Selection Allowed	rowSelectionAllowed	Behavior	boolean	This flag is used in conjunction with the Column Selection Allowed flag to

Name	Scripting	Category	Property Type	Description
				determine whether not whole-rows, whole-columns, or both.
Header Visible	headerVisible	Appearance	boolean	Whether or not the table header is visible.
Resizing Allowed	resizingAllowed	Behavior	boolean	Whether or not the user is allowed to resize table headers or not.
Row Height	rowHeight	Appearance	int	The height of each row, in pixels.
Odd Row Background	oddBackground	Appearance	Color	The color which odd rows will be colored if background mode is 'Alternating'.
Selection Background	selectionBackground	Appearance	Color	The background color of a selected cell.
Selection Foreground	selectionForeground	Appearance	Color	The foreground color of a selected cell.
Show Horizontal Grid Lines?	showHorizontalLines	Appearance	boolean	Displays horizontal gridlines making it easier to read.
Show Vertical Grid Lines?	showVerticalLines	Appearance	boolean	Displays vertical gridlines making it easier to read.

Name	Scripting	Category	Property Type	Description
Grid Line Color	gridColor	Appearance	Color	The color used to draw grid lines.
Column Attributes Data	columnAttributesData	Data	Dataset	The dataset describing the column attributes.
Selected Row	selectedRow	Data	int	The index of the first selected row, or -1 if none.

Scripting

Scripting Functions

activateRecipe

- Description

This script will activate the current recipe name on the selected item path using the Manual Values entered in the table.

- Parameters

None

- Return

A String representing any error that was encountered. The string will be blank if the recipe was successfully activated.

- Scope

Client

Code Examples

Code Snippet

```
#Sample script to activate the currently viewed recipe:

recipeEditorTable = event.source.parent.getComponent('Rec
ipe Editor Table')
result = recipeEditorTable.activateRecipe()
if result != '':
 system.gui.messageBox(result, "Recipe activation error")
```

updateLiveValues

- Description

This script will update the live values of the recipe referenced tags in the table.

- Parameters

None

- Return

A String representing any error that was encountered. The string will be blank if the recipe was successfully activated.

- Scope

Client

saveRecipe

- Description

This script will update the settings for the recipe with the current live recipe values.

- Parameters

None

- Return

A String representing any error that was encountered. The string will be blank if the recipe was successfully activated.

- Scope

Client

reset

- Description

This script will reset the manual values to the recipe settings value.

- Parameters

None

- Return

None

- Scope

Client

Extension Functions

getBackgroundAt

- Description

Called for each cell, returns the appropriate background color. Do not block, sleep, or execute any I/O; called on painting thread.

- Parameters

self - A reference to the component that is invoking this function.

row - The row index of the cell.

col - The column index of the cell.

isSelected - A boolean representing if this cell is currently selected.

value - The value in the table's dataset at index [row, col]

defaultColor - The color the table would have chosen if this function was not implemented.

- Return

Default color

- Scope

Client

getForegroundAt

- Description

Called for each cell, returns the appropriate foreground (text) color. Do not block, sleep, or execute any I/O; called on painting thread.

- Parameters

self - A reference to the component that is invoking this function.

row - The row index of the cell.

col - The column index of the cell.

isSelected - A boolean representing if this cell is currently selected.

value - The value in the table's dataset at index [row, col]

defaultColor - The color the table would have chosen if this function was not implemented.

- Return

Default color

- Scope

Client

getDisplayTextAt

- Description

Called for each cell, returns a String which will be used as the text of the cell. Do not block, sleep, or execute any I/O; called on painting thread.

- Parameters

self - A reference to the component that is invoking this function.

row - The row index of the cell.

col - The column index of the cell.

isSelected - A boolean representing if this cell is currently selected.

value - The value in the table's dataset at index [row, col]

defaultText - The string the table would have chosen if this function was not implemented.

- Return

Default text

- Scope

Client

filterRow

- Description

Called for each cell, returns a Boolean indicating if this row is to be filtered from view. Do not block, sleep, or execute any I/O; called on painting thread.

- Parameters

self - A reference to the component that is invoking this function.

row - The row index in the dataset.

unfilteredData - The dataset before filtering.

- Return

Boolean

- Scope

Client

Event Handlers

cell

cellEdited

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it.

Property	Description
source	The component that fired this event.
oldValue	The old value in the cell that changed.
newValue	The new value in the cell that changed.
row	The row of the dataset this cell represents.
column	The column of the dataset this cell represents.

focus

focusGained

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it.

Property	Description
source	The component that fired this event.
oppositeComponent	

Property	Description
	The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vise versa.

focusLost

This event occurs when a component that had the input focus lost it to another component.

Property	Description
source	The component that fired this event.
oppositeComponent	The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vise versa.

key

keyPressed

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.

Property	Description
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

keyReleased

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

keyTyped

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.

Property	Description
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.

Property	Description
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.

Property	Description
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	

Property	Description
	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

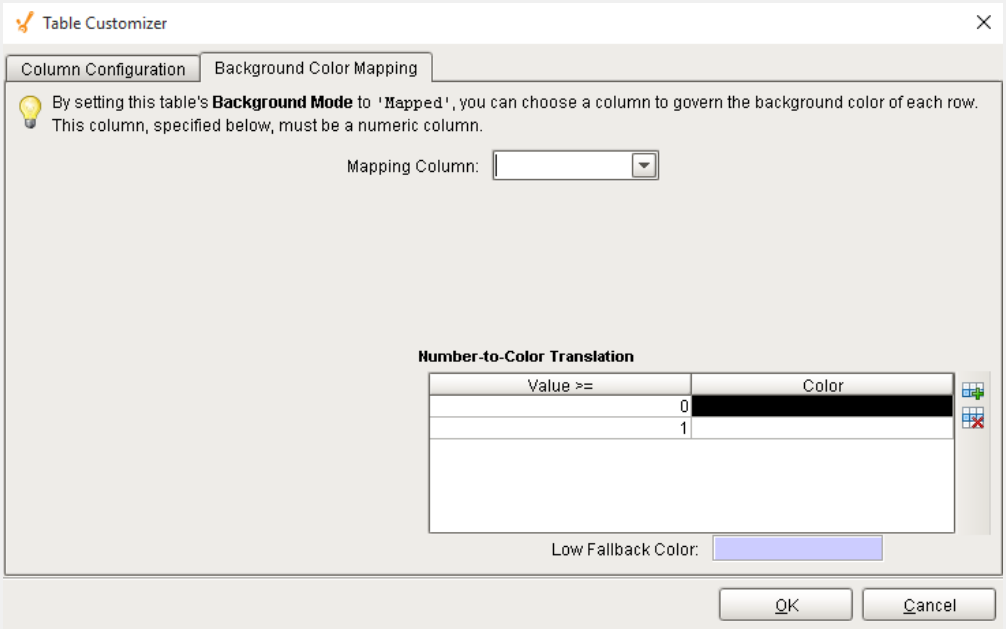
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

Table Customizer

Table Customizer manages the data entered into the Recipe Editor Table.



Examples

To use the recipe table you must supply an item path and recipe name then the table will populate with the related values.

It has the capability for end users to do the following recipe related tasks:

- View the current tag values related to this recipe and equipment path. This can be set to update at a specific interval.
- Manually adjust the recipe value before writing it down to the tags.
- Update the selected recipe with the current live values.

Name	Recipe Setting	Live Value	Manual Value
int2_RV	1	0	1
boolean_RV	true	false	true
int1_RV	121	0	121
int8_RV	0	0	0
BRecipe	777	56	777
datetime_RV	Wed Dec 31 16:00:00 PST 1969	Fri May 16 12:00:00 PDT 2014	Wed Dec 31 16:00:00 PST 1969
float4_RV	55.9	8.25	55.9
int4_RV	0	0	0
CRecipe(12)	888	88	888
DRecipe(1)	999	9	999
string_RV	This is a String	This ® is a String	This is a String
Alpha Recipe	666	14	666

RecipeEditorTable_Main

The table coulmns displayed can be customized by using the standard Ignition table customizer available by right clicking on the table component in the designer. The available fields are:

- Name

The recipe value name

- Recipe Setting

The set value assigned in the recipe

- Live Value

The current live tag value referenced by this recipe value

- Manual Value

A value that is manually set by the user. You must call a method of the component in order to have this value affect the recipe or the tag value.

- Description

The description of this recipe value

- Units

The units of this recipe value

- Assigned By

The assigned by name of this recipe value

- Format

The format of this recipe value

- dataType

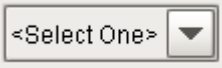
The data type of this recipe value

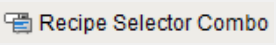
- tagPath

The tag path referenced by this recipe value

Recipe Selector Combo

General



Component Palette Icon: 

Description

A component that is added to Ignition windows to select recipes in a drop down list. This is just one method of selecting a recipe for a production line, cell, cell group or location. For more information on the other methods see the Selecting Recipes section. The Recipe Selector Combo component is automatically updated when recipes are added, removed, etc.

Properties

Name	Scripting	Category	Property Type	Description
Item Path	itemPath	Data	String	The path of the production item.

Name	Scripting	Category	Property Type	Description
Recipe Name Filter	recipeNameFilter	Data	String	Optionally, limit changelog results to recipe name.
Selected Recipe Name	selectedRecipeName	Data	String	The name of the currently selected recipe.
Styles	styles	Appearance	DataSet	Contains the component's styles.
Horizontal Alignment	horizontalAlignment	Layout	int	Determines the alignment of the contents along the X axis.
Vertical Alignment	verticalAlignment	Layout	int	Determines the alignment of the contents along the Y axis.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

focus

focusGained

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it.

Property	Description
source	The component that fired this event.
oppositeComponent	The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vise versa.

focusLost

This event occurs when a component that had the input focus lost it to another component.

Property	Description
source	The component that fired this event.
oppositeComponent	The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vise versa.

key

keyPressed

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is.

Property	Description
source	The component that fired this event.
keyCode	

Property	Description
	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

keyReleased

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This

Property	Description
	provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

keyTyped

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.

shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	**Description**
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	**Description**
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	

Property	Description
	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

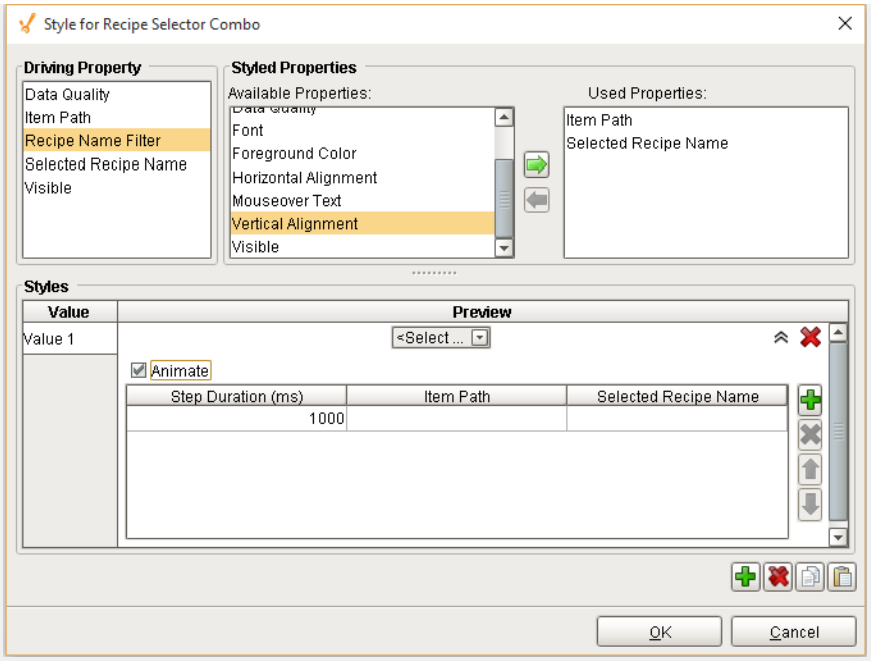
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.

Property	Description
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

Customizers

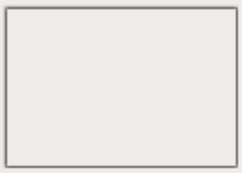
This component does not have a customizer however this component relies on custom styles. The example below has the styles defined here:



Examples

Recipe Selector List

General



Component Palette Icon: Recipe Selector List

Description

A component that is added to Ignition windows to select recipes in a scrollable list. This is just one method of selecting a recipe for a production line, cell, cell group or location. For more information on the other methods see the Selecting Recipes section. The Recipe Selector List component is automatically updated when recipes are added, removed, etc. The current recipe selected is also automatically updated if the recipe changes from a different source.

To select a recipe for a production item, the user can right click on a recipe then select the Select Recipe menu item. To cancel a recipe, right click on the selected recipe then select the Cancel Recipe menu item. This component also supports additional menu item to be added by using the User Menu Items property and the userMenuItemClicked event.

Properties

Name	Scripting	Category	Property Type	Description
Read Only	readOnly	Behavior	Boolean	If true, then user cannot select or cancel recipes.
Item Path	itemPath	Data	String	The path of the production item.
Recipe Name Filter	recipeNameFilter	Data	String	Optionally, limit changelog results to recipe name.
Selected Icon Path	selectedIconPath	Appearance	String	The relative path of the selected icon image.
Icon-Text Spacing	iconTextGap	Appearance	int	The space.
Scroll Bar Width	scrollBarWidth	Appearance	int	The width of the scroll bars. If zero, use the system default scroll bar width.
	userMenuItems	Behavior	DataSet	

Name	Scripting	Category	Property Type	Description
User Menu Items				A dataset that stores user menu items.
Selected Recipe Name	selectedRecipeName	Data	String	The name of the currently selected recipe.
Styles	styles	Appearance	DataSet	Contains the component's styles.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

focus

focusGained

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it.

Property	Description
source	The component that fired this event.
oppositeComponent	The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vise versa.

focusLost

This event occurs when a component that had the input focus lost it to another component.

Property	Description
source	The component that fired this event.
oppositeComponent	The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vise versa.

key

keyPressed

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This

Property	Description
	provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

keyReleased

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.

shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

keyTyped

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	**Description**
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

Property	Description
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

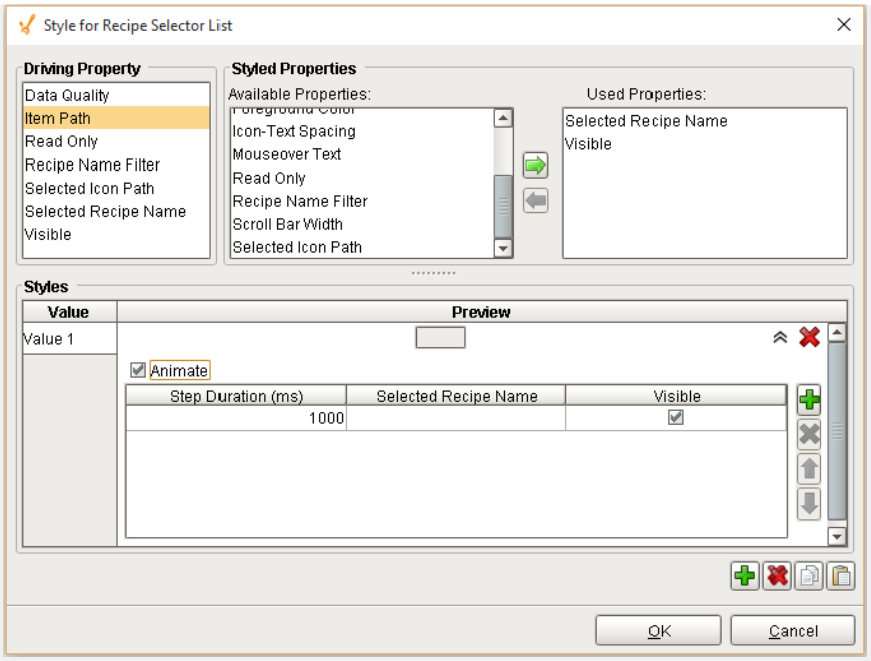
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.

Property	Description
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

Customizers

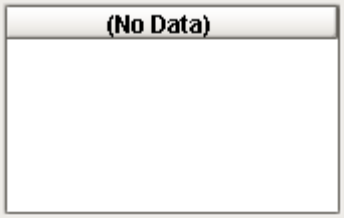
This component does not have a customizer however this component relies on custom styles. The example below has the styles defined here:



Examples

Recipe Variance Viewer

General



Component Palette Icon: Recipe Variance Viewer

Description

A component that is added to Ignition windows to display recipe variances in a table. This is just one method of viewing a recipe variances and for more information on the other methods see the Variance Monitoring section. The Recipe Variance Viewer is

automatically updated when live recipe value variances are detected by the Recipe / Changeover Module. The Show Full Details property will cause the initial values when the recipe was selected, to also be included. This component simplifies displaying recipe variances by handling all of the backend database queries based on the property settings of the component.

Properties

Name	Scripting	Category	Property Type	Description
Start Date	startDate	Data	DateTime	Start Date.
End Date	endDate	Data	DateTime	End Date.
Recipe Name Filter	recipeNameFilter	Data	String	Optionally, limit recipe variance results to a recipe name.
Sub Recipe Name Filter	subRecipeNameFilter	Data	String	Optionally, limit recipe variance results to a sub recipe name.
Item Path Filter	itemPathFilter	Data	String	Optionally, limit recipe variance results to a production item path.
Value Name Filter	recipeValueNameFilter	Data	String	Optionally, limit recipe variance results to recipe value name.
Recipe TrackingUUID	recipeTrackingUUID	Data	String	

Name	Scripting	Category	Property Type	Description
				Optionally, limit recipe variance results to recipe trackingUUID.
Show Full Details	showFullDetails	Data	Boolean	If true, show initial recipe values and any changes, else only show deviations from the initial recipe values.
Data	data	Data	DataSet	The data for the table.
Header Font	headerFont	Appearance	Font	Font of the table's header text.
Header Foreground Color	headerForeground	Appearance	Color	The foreground color of the table's header.
Row Selection Allowed	rowSelectionAllowed	boolean	boolean	This flag is used in conjunction with the Column Selection Allowed flag to determine whether not whole-rows, whole-columns, or both.
Header Visible	headerVisible	Appearance	boolean	Whether or not the table header is visible.
Resizing Allowed	resizingAllowed	Behavior	boolean	

Name	Scripting	Category	Property Type	Description
				Whether or not the user is allowed to resize table headers or not.
Row Height	rowHeight	Appearance	int	The height of each row, in pixels.
Odd Row Background	oddBackground	Appearance	Color	The color which odd rows will be colored if background mode is 'Alternating'.
Selection Background	selectionBackground	Appearance	Color	The background color of a selected cell.
Selection Foreground	selectionForeground	Appearance	Color	The foreground color of a selected cell.
Show Horizontal Grid Lines?	showHorizontalLines	Appearance	boolean	Displays horizontal gridlines making it easier to read.
Show Vertical Grid Lines?	showVerticalLines	Appearance	boolean	Displays vertical gridlines making it easier to read.
Grid Line Color	gridColor	Appearance	Color	The color used to draw grid lines.
Column Attributes Data	columnAttributesData	Data	DataSet	The dataset describing the column attributes.

Name	Scripting	Category	Property Type	Description
Selected Row	selectedRow	Data	int	The index of the first selected row, or -1 if none.

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

mouse

mouseClicked

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.

Property	Description
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseEntered

This event fires when the mouse enters the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	

Property	Description
	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseExited

This event fires when the mouse leaves the space over the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.

Property	Description
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mousePressed

This event fires when a mouse button is pressed down on the source component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMotion

mouseDragged

Fires when the mouse moves over a component after a button has been pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.

Property	Description
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

mouseMoved

Fires when the mouse moves over a component, but no buttons are pushed.

Property	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.

Property	Description
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Ctrl key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

propertyChange

propertyChange

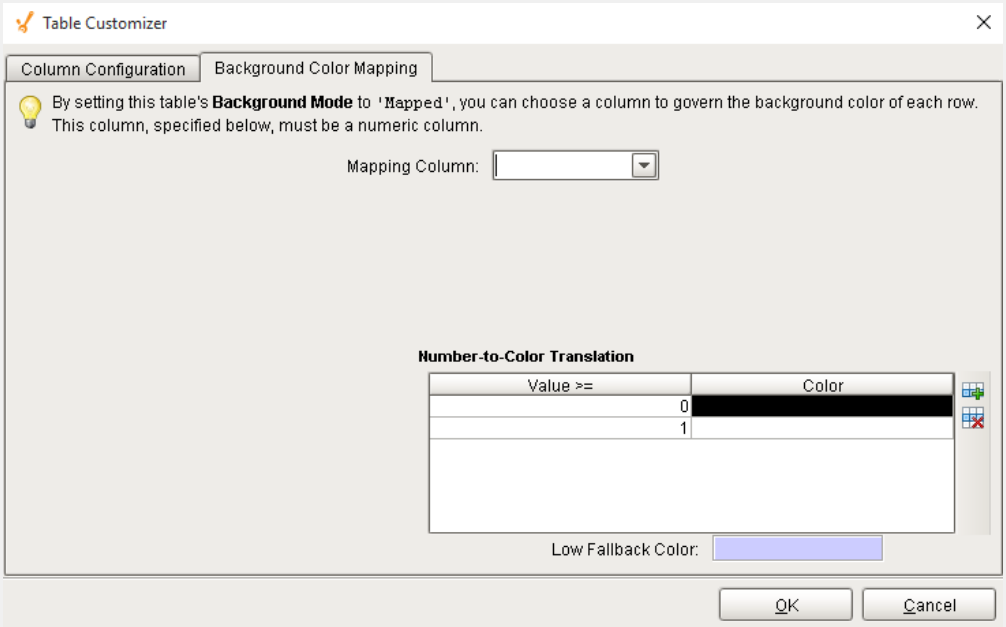
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Property	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Customizers

Table Customizer

Table Customizer manages the data entered into the Recipe Variance Viewer.



Examples

9.5.6 InstrumentInterface Components

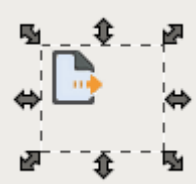
This section is a reference for all of the components that come with the Instrument Interface Module. The components are displayed during design time, but are not visible during runtime. There are two types of components:

1. File Monitor
2. Serial Controller

File Monitor handles the functions for reading data in files, and Serial Controller handles the functions for device communications.

File Monitor Controller

General



Component Palette Icon: File Monitor Controller

Not intended for very large files. Parsed Files are loaded into memory.

Description

An invisible component that handles detecting, reading and parsing functions to provide reading data in files. The term invisible component means that this component appears during design time, but is not visible during runtime.

Info

In design time, the last raw data read from a file can be sent to the selected template defined by the Instrument Interface Name by right clicking on the component in the Ignition designer and selecting the Send to Template menu item. This will also select and display the template and replace the existing textual data with the last raw data read.

If the Enable Monitoring property is selected and the designer is preview mode or client has the window open that contains a file monitor component, this component will actively look for files to process. The files that it will process are specified by the File Path property and can contain wildcard characters.

This component will perform a test lock on the file prior to processing to insure that writing to the file is complete. This prevents processing a file before it is ready. This is an important feature, if processing of a file starts and data is still being written to the file it will wither cause errors or incomplete data will be processed.

Properties

Name	Scripting	Category	Property Type	Description
Move To Directory Path	moveToDirectoryPath	File Path Config	String	If After Processing Handling is set to Move File, the directory to move processed file to.
Instrument Interface Name	instrumentInterfaceName	Monitor File Config	String	The name of the instrument interface configuration name to use.
File Name Date Format	fileNameDateFormat	Monitor File Config	String	The date format used when determining the order to process files.
Encoding	encoding	Monitor File Config	String	Character encoding.
Enable Monitoring	enableMonitoring	Monitor File Config	Boolean	If true, presents of files will be monitored.
Monitor Rate	monitorRate	Monitor File Config	int	Interval in milliseconds to monitor file existence.
Last File Read At	lastFileReadAt	Status	DateTime	Date and time the last file was read at.
Last File Processed	lastFileProcessed	Status	String	Name and path of the last file processed.
Error Message	errorMessage	Status	String	Error message.

Scripting Functions

read

- Description

Check existence of and process one files. If multiple files exist only one file is processed because the ParseResults are returned.

- Parameters

None

- Return

ParseResults - Returns a ParseResults object containing all the values that were parsed from the raw data. See ParseResults object reference for more information about reading values from the ParseResults object.

- Scope

Client

read(fileName)

- Description

Check existence of and process one files. If multiple files exist only one file is processed because the ParseResults are returned.

- Parameters

String fileName - File path to file to process if it exists.

- Return

ParseResults - Returns a ParseResults object containing all the values that were parsed from the raw data. See ParseResults object reference for more information about reading values from the ParseResults object.

- Scope

Client

parseText(template, text)

- Description

Parses the given text by using the template of templateName.

- Parameters

String templateName - The template to use for parsing the text.

String text - The text to be parsed.

- Return

ParseResults See ParseResults object for information about accessing parsed values contained in the parse results.

- Scope

Client

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

monitorFile

onError

Is fired when an error occurs during reading file contents. The errorMessage property can be read to get the error message.

Property	Description
source	The component that fired this event.
errorMessage	Is fired when an error occurs on the the serial communication port. The errorMessage property can be read to get the error message.

parse

onBeforeParse

Is fired before raw data is sent to the parsing engine to be parsed. This provides a method for the raw data to be modified before being parsed. It can be useful to remove unwanted characters or merging more data into the raw data before parsing.

Property	Description
source	The component that fired this event.
data	Modified data to send to the parsing engine.

onAfterParse

Is fired after the raw data has been parsed.

Property	Description
source	The component that fired this event.
parseResults	Object containing all the values that were parsed from the raw data.
data	Parsed data.

Code Snippets

```
parseResults = event.getParseResults()
if parseResults.isValid():
 event.source.parent.getComponent('LabelDate').text = str(parse
Results.getValue("Date"))
 event.source.parent.getComponent('LabelTime').text = str(parse
Results.getValue("Time"))
 event.source.parent.getComponent('LabelSampleNo').text = str(p
arseResults.getValue("Sample No"))
 event.source.parent.getComponent('LabelAlcohol').text = str(pa
rseResults.getValue("Alcohol"))
 event.source.parent.getComponent('LabelDensity').text = str(pa
rseResults.getValue("Density"))
 event.source.parent.getComponent('LabelCalories').text = str(p
arseResults.getValue("Calories"))
```

Customizers

This component does not have any custom properties.

Example 1

File monitor component can be used to move a file to a specific location. For this, first give the current location of the file in the **File Path** property of the component. Now specify the destination location in **Move To Directory Path** property.

Property Name	Value
File Path	C:\Users\sarah\Downloads\aaa.txt
Move To Directory Path	C:\Users\sarah\Desktop\
File Processing Priority	0
After Processing Handling	1

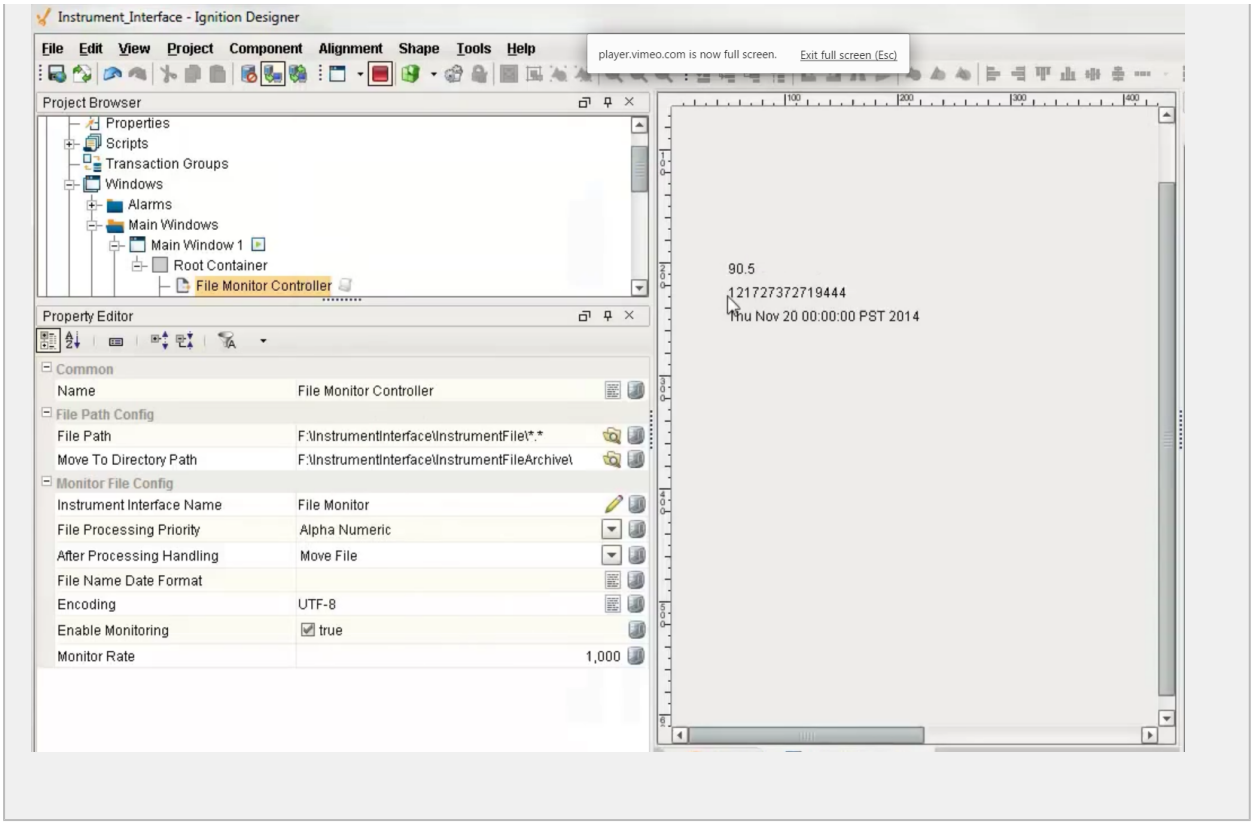
Example 2

File monitor component can be used to parse a file.

For this, right click on the component and hit scripting. Select the **OnAfterParse** event handler and click on the Script Editor tab. Copy the following script so that the contents inside the parsed text file is rendered with the label component.

```
result = event.getParseResults()
title = result.getValue('PositionModeBox')
timeStamp = result.getValue('DateLabel')
scaleID = result.getValue('ScaleLabel')
event.source.parent.getComponent('Label').text = str(title)
event.source.parent.getComponent('Label 1').text = str(scaleID)
event.source.parent.getComponent('Label 2').text = str(timeStam
p)
```

In the preview mode, the Ignition designer appears as the following.

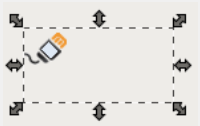


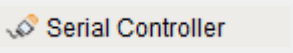
In design time, the last raw data read from a file can be sent to the selected template defined by the Instrument Interface Name by right clicking on the component in the Ignition designer and selecting the Send to Template menu item. This will also select and display the template and replace the existing textual data with the last raw data read.

If the Enable Monitoring property is selected and the designer is preview mode or client has the window open that contains a file monitor component, this component will actively look for files to process. The files that it will process are specified by the File Path property and can contain wildcard characters.

Serial Controller

General



Component Palette Icon: 

Description

An invisible component that handles serial communications and parsing functions to provide instrument device communications. The term invisible component means that this component appears during design time, but is not visible during runtime. The serial component helps to send and receive information from RS-232 devices.

In design time, the last raw data received from the communication port can be sent to the selected template defined by the Instrument Interface Name by right clicking on the component in the Ignition designer and selecting the Send to Template menu item. This will also select and display the template and replace the existing textual data with the last raw data received.

In run time, if the Instrument Interface Name property is set, raw data received from the serial communications port will be sent to the parsing engine on the gateway to be parsed. The template used to parse the raw data is named the same as the value of the Instrument Interface Name property.

Properties

Name	Scripting	Category	Property Type	Description
Port	portName	Port Config	String	The name of the system serial port to use.
Baud Rate	baudRate	Port Config	int	Serial communication baud rate.
Auto Open Port	autoOpen	Port Config	Boolean	If true, automatically open the serial port.
Instrument Interface Name	instrumentInterfaceName	Port Config	String	The name of the instrument interface configuration name to use.
	clearBufferBeforeSend		Boolean	

Name	Scripting	Category	Property Type	Description
Clear Buffer Before Send		Port Config		Clear the receive buffer before sending data.
Correct CRLF	correctCRLF	Port Config	Boolean	Coorect any combination of end of line characters to carriage return.
Default Read Timeout	defaultReadTimeout	Port Config	int	The default number of milliseconds to wait while reading.
Encoding	encoding	Request Handling	String	Character encoding.
Unsolicited Requests	unsolicitedRequests	Request Handling	Boolean	If true, accept unsolicited requests from the device.
Enable Polled Requests	enablePolledRequests	Request Handling	Boolean	If true, requests are made at a fixed timed interval.
Polling Rate	pollingRate	Request Handling	int	Interval in milliseconds to issue poll request.
Enable Capture	enableCapture	Logging	Boolean	If true, write all received and sent data to a capture file.
Capture File Path	captureFilePath	Logging	String	The file path on the local computer to create the capture file.

Serial Module Loaded	serialModuleLoaded	Port Status	Boolean	If true, the serial module has been installed and is loaded.
Serial Port Open	serialPortOpen	Port Status	Boolean	If true, the serial port is open.
Last Data Sent At	lastDataSentAt	Port Status	DateTime	Date and time of last data transmission.
Last Data Received At	lastDataReceivedAt	Port Status	DateTime	Date and time of last data received.
Error Message	errorMessage	Port Status	String	Error message.

Scripting

Scripting Functions

openPort

- Description

Attempts to open the port. If an error occurs the errorMessage property will be set and an exception will be thrown.

- Parameters

None

- Return

Nothing

- Scope

Client

closePort

- Description

Attempts to close the port. If an error occurs the errorMessage property will be set and an exception will be thrown.

- Parameters

None

- Return

Nothing

- Scope

Client

writeString

- Description

Write value of the text parameter to the communication port. If an error occurs the errorMessage property will be set and an exception will be thrown.

- Parameters

String text - The text to write to the port.

- Return

Nothing

- Scope

Client

writeBytes

- Description

Write value of the data parameter to the communication port. If an error occurs the errorMessage property will be set and an exception will be thrown.

- Parameters

byte[] - The byte array to write to the port.

- Return

Nothing

- Scope

Client

readString

- Description

Reads and returns string data from the communication port. If an error occurs the errorMessage property will be set and an exception will be thrown. If no data is received within the default timeout setting, then an empty string will be returned.

- Parameters

None

- Return

String data - The data read from the port.

- Scope

Client

readString(timeout)

- Description

Reads and returns string data from the communication port. If an error occurs the errorMessage property will be set and an exception will be thrown. If no data is received within the value specified in the timeout parameter, then an empty string will be returned.

- Parameters

Integer timeout - The time in milliseconds to wait for a response from the port.

- Return

String data - The data read from the port.

- Scope

Client

readBytes(count)

- Description

Reads and returns byte array data from the communication port. If an error occurs the errorMessage property will be set and an exception will be thrown. If the number of characters specified by the count parameter are not received within the default timeout setting, then any characters received will be returned.

- Parameters

Integer count - The number of bytes to read from the port.

- Return

byte[ ] The data read from the port.

- Scope

Client

readBytes(count, timeout)

- Description

Reads and returns byte array data from the communication port. If an error occurs the errorMessage property will be set and an exception will be thrown. If the number of characters specified by the count parameter are not received within the value specified in the timeout parameter, then any characters received will be returned.

- Parameters

Integer count - The number of bytes to read from the port.

Integer timeout - The time in milliseconds to wait for a response from the port.

- Return

byte[ ] The data read from the port.

- Scope

Client

readUntil(delimiter, includeDelimiter)

- Description

Reads and returns string data from the communication port up until the character specified by the delimiter parameter. If an error occurs the errorMessage property will be set and an exception will be thrown. If the delimiter character is not received within the default timeout setting, then any characters received will be returned.

- Parameters

Char delimiter - The delimiter to read until.

- Return

Boolean includeDelimiter - If true the delimiter will be included in the return value.

- Scope

Client

readUntil(delimiter, includeDelimiter, timeout)

- Description

Reads and returns string data from the communication port up until the character specified by the delimiter parameter. If an error occurs the errorMessage property will be set and an exception will be thrown. If the delimiter character is not received within the value specified in the timeout parameter, then any characters received will be returned.

- Parameters

Char delimiter - The delimiter to read until.

Boolean includeDelimiter - If true the delimiter will be included in the return.

Integer timeout - The time in milliseconds to wait for a response from the port.

- Return

byte[ ] The data read from the port.

- Scope

Client

clearBuffer

- Description

Clear all data existing in the communication port receive buffer. If an error occurs the errorMessage property will be set and an exception will be thrown.

- Parameters

None

- Return

Nothing

- Scope

Client

isSerialSupported

- Description

Determines if the client serial module is loaded and available to be used with this component.

- Parameters

None

- Return

Boolean True if serial support is available.

- Scope

Client

parseText

- Description

Parses the given text by using the template of templateName.

- Parameters

String templateName - The template to use for parsing the text.

String text - The text to be parsed.

- Return

ParseResults - A ParseResults object (see ParseResults object for information about accessing parsed values contained in the parse results.)

- Scope

Client

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

parse

onBeforeParse

Is fired before raw data is sent to the parsing engine to be parsed. This provides a method for the raw data to be modified before being parsed. It can be useful to remove unwanted characters or merging more data into the raw data before parsing.

Property	Description
source	The component that fired this event.
data	Modified data to send to the parsing engine.

onAfterParse

Is fired after the raw data has been parsed.

Property	Description
source	The component that fired this event.
parseResults	Object containing all the values that were parsed from the raw data.

Property	Description
data	Parsed data.

Code Snippet

```
results = event.getParseResults()
if results != None and results.isRequiredValid():
 reading = results.getValue("reading")
 event.source.parent.getComponent('Numeric Text Field').
doubleValue = reading.getValue()
 event.source.parent.getComponent('Sample Entry').
populateMeasurement("Viscosity", reading.getValue(), 1)
else:
 system.gui.messageBox("Error reading value from
instrument.")
```

serialPort

onOpen

Is fired when the serial communication port is opened.

Property	Description
source	The component that fired this event.
errorMessage	Is fired when an error occurs on the the serial communication port. The errorMessage property can be read to get the error message.
stringData	The string data read from the port.
byteData	The byte data read from the port.

onClose

Is fired when the serial communication port is closed.

Property	Description
source	The component that fired this event.
errorMessage	

Property	Description
	Is fired when an error occurs on the the serial communication port. The errorMessage property can be read to get the error message.
stringData	The string data read from the port.
byteData	The byte data read from the port.

onSend

Is fired when data has been sent to the port.

Property	Description
source	The component that fired this event.
errorMessage	Is fired when an error occurs on the the serial communication port. The errorMessage property can be read to get the error message.
stringData	The string data read from the port.
byteData	The string data read from the port.

onReceive

Is fired when data has been received from the serial communication port.

Property	Description
source	The component that fired this event.
errorMessage	Is fired when an error occurs on the the serial communication port. The errorMessage property can be read to get the error message.
stringData	The string data read from the port.
byteData	The byte data read from the port.

onPoll

Is fired when the serial communications port has been polled for data .

Property	Description
source	The component that fired this event.
errorMessage	Is fired when an error occurs on the the serial communication port. The errorMessage property can be read to get the error message.
stringData	The string data read from the port.
byteData	The byte data read from the port.

onError

Is fired when an error occurs on the the serial communication port. The errorMessage property can be read to get the error message.

Property	Description
source	The component that fired this event.
errorMessage	Is fired when an error occurs on the the serial communication port. The errorMessage property can be read to get the error message.
stringData	The string data read from the port.
byteData	The byte data read from the port.

Customizers

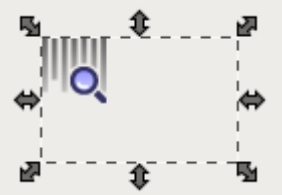
This component does not have any custom properties.

Examples

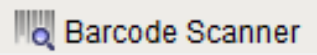
9.5.7 BarcodeScanner Components

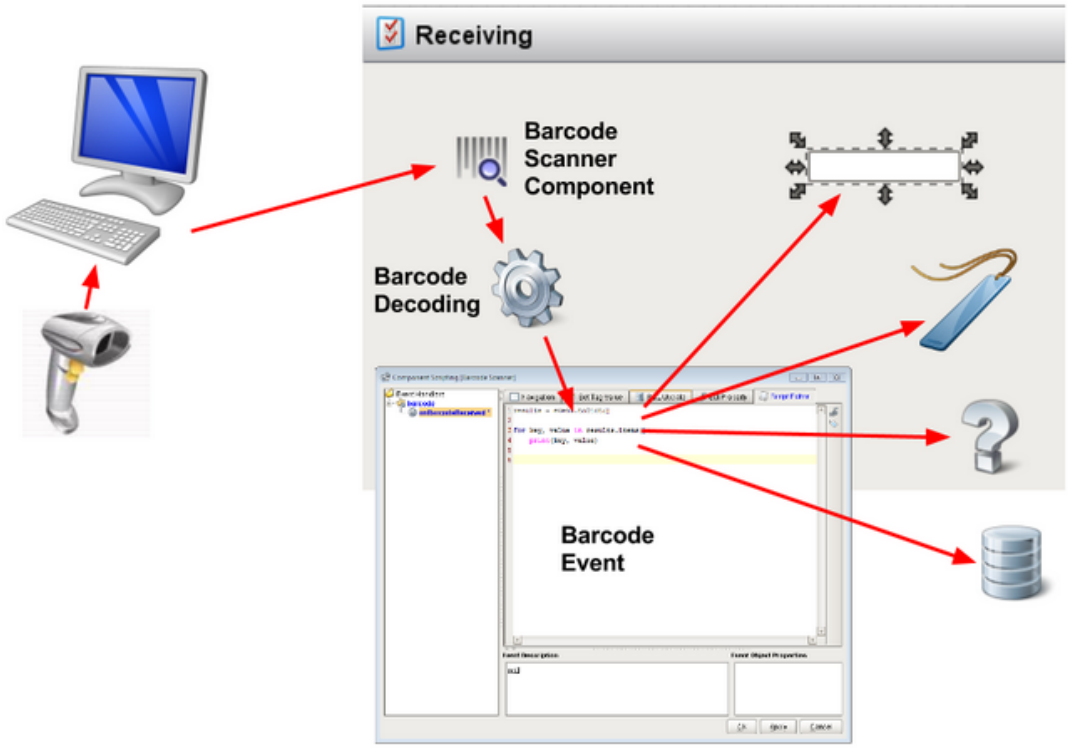
BarcodeScanner

General



Component Palette Icon:





Barcode Detection and Decoding

Description

An invisible component that listens for barcode input received via a keyboard interface. The term invisible component means that this component appears during design time, but is not visible during runtime.

There are over 100 barcode patterns predefined to support the most common and GS1 international standards. All the patterns are customizable and can be enabled as required. Multiple components can be used on the same window to detect different types of barcodes.

The component listens to keystrokes based on a defined preamble and an optional post amble specification, then decodes the raw barcode using regular expression patterns and passes the results to a script event. The component is designed to work in the background and allow scripting to process the results of the barcode content, such as auto fill fields or update information.

Info

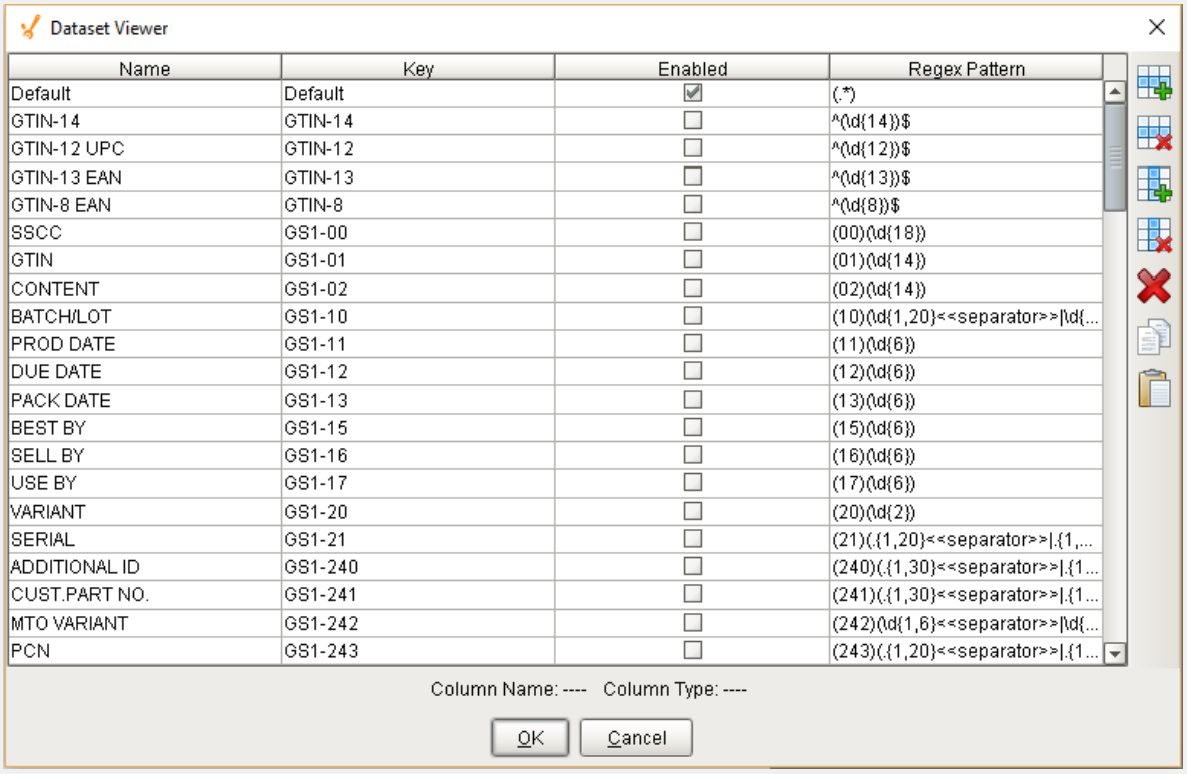
This component when enabled will listen for barcode input only on Stage or Published clients. It doesn't activate in the designer's preview mode.

Properties

Name	Scripting	Category	Property Type	Description
Barcode Pattern Configuration	patternConfiguration	Barcode	DataSet	The set of Regex barcode patterns to search for a match.
Decode Method	decodeMethod	Barcode	int	Determines if the decoding of the barcode does a single-pass or consume search method for matching patterns.
Preamble	preamble	Barcode	String	A character, string, or regex unicode value representing the preamble pattern.

Name	Scripting	Category	Property Type	Description
Postamble	postamble	Barcode	String	A character, string, or regex unicode value representing the postamble pattern.
Separator	separator	Barcode	String	A character, string, or regex unicode value representing the barcode group separator pattern.
Read Timeout	readTimeout	Barcode	int	The pause timeout between keystrokes to check for a barcode scan.

Barcode Dataset viewer



Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

barcode

onBarcodeReceived

Is fired after the component decodes a raw barcode. It is passed a BarcodeEvent object that contains the results of the decoding, any error message, and the raw barcode.

Property	Description
source	The component that fired this event.
results	Decoded barcode results that matched the patterns given.

Example 1

```
Sample #1 onBarcodeReceived event script:
#Sample Use of the Python Dictionary method to get barcode
values from a single-pass #decode method
results = event.toDict()                # converts Java
hashtable to Python dictionary
#Sample method to print out all result values
for key, value in results.items():
    print(key, value)
#Look for an error message and display it to text box
if event.hasErrorMessage():
 event.source.parent.getComponent('txtError').text = event.
getErrorMessage()
```

```
 system.util.beep()
else:
 #Update screen components from decode results
 if 'GTIN-12' in results:
        event.source.parent.getComponent('txtUPC').text =
results['GTIN-12'][0]
 if 'GTIN-13' in results:
        event.source.parent.getComponent('txtUPC').text =
results['GTIN-13'][0]
 if 'GTIN-14' in results:
        event.source.parent.getComponent('txtUPC').text =
results['GTIN-14'][0]
```

Example 2

```
Sample #2 onBarcodeReceived event script:
# get results using Java Hashtable
results = event.getResults()
if event.hasErrorMessage():
 event.source.parent.getComponent('txtError').text = event.
getErrorMessage()
 system.util.beep()
if results.containsKey('GS1-01'):
 event.source.parent.getComponent('txtGTIN').text = results.
get('GS1-01').get(1)
if results.containsKey('GS1-10'):
 event.source.parent.getComponent('txtBatchLot').text =
results.get('GS1-10').get(1)
if results.containsKey('GS1-17'):
 event.source.parent.getComponent('calUseBy').date = event.
GS1ConvertToDate(results.get('GS1-17').get(1))
if results.containsKey('GS1-310'):
 event.source.parent.getComponent('numNetWeight').
floatValue = event.GS1ConvertToFloat(results.get('GS1-310').
get(2), results.get('GS1-310').get(1))
if results.containsKey('GS1-390'):
 event.source.parent.getComponent('numAmount').doubleValue =
 event.GS1ConvertToDouble(results.get('GS1-390').get(2),
results.get('GS1-390').get(1))
if event.hasUnmatched:
 event.source.parent.getComponent('txtOutput').text = event.
getUnmatched()
```

Customizers

This component does not have any custom properties.

Examples

9.6 Objects

Objects are at the heart of the Sepasoft Product Suite. They provide the methods and attributes that all our components interact with to create production schedules, start runs, enforce production control and analyze production results.

We have objects for each of the products as well as some objects that are common across all products. In future releases, almost all products will be based on the MES objects.

9.6.1 Object UUIDs

Each MES object and even other properties or data has to be uniquely identified. The typically method of doing this is to use an identification number generated by the database, which an integer, starting at 1 and increasing over time. But, this method becomes a problem when data from multiple systems, that each have their own database that are generating identification numbers, is push up to an enterprise or higher level system. In the database of the enterprise or higher level system, the identification numbers will have duplicates which causes inconsistencies of historical data.

For this reason, the Sepasoft MES system uses Universally Unique Identifiers (UUID). A UUID represents a 128-bit value that enables distributed systems to uniquely identify information without significant central coordination. Information identified with a UUID can therefore be combined into a single database without needing to be concerned about duplicates.

Even though true UUIDs are a data type of their own, all UUID values within the Sepasoft MES modules are strings.

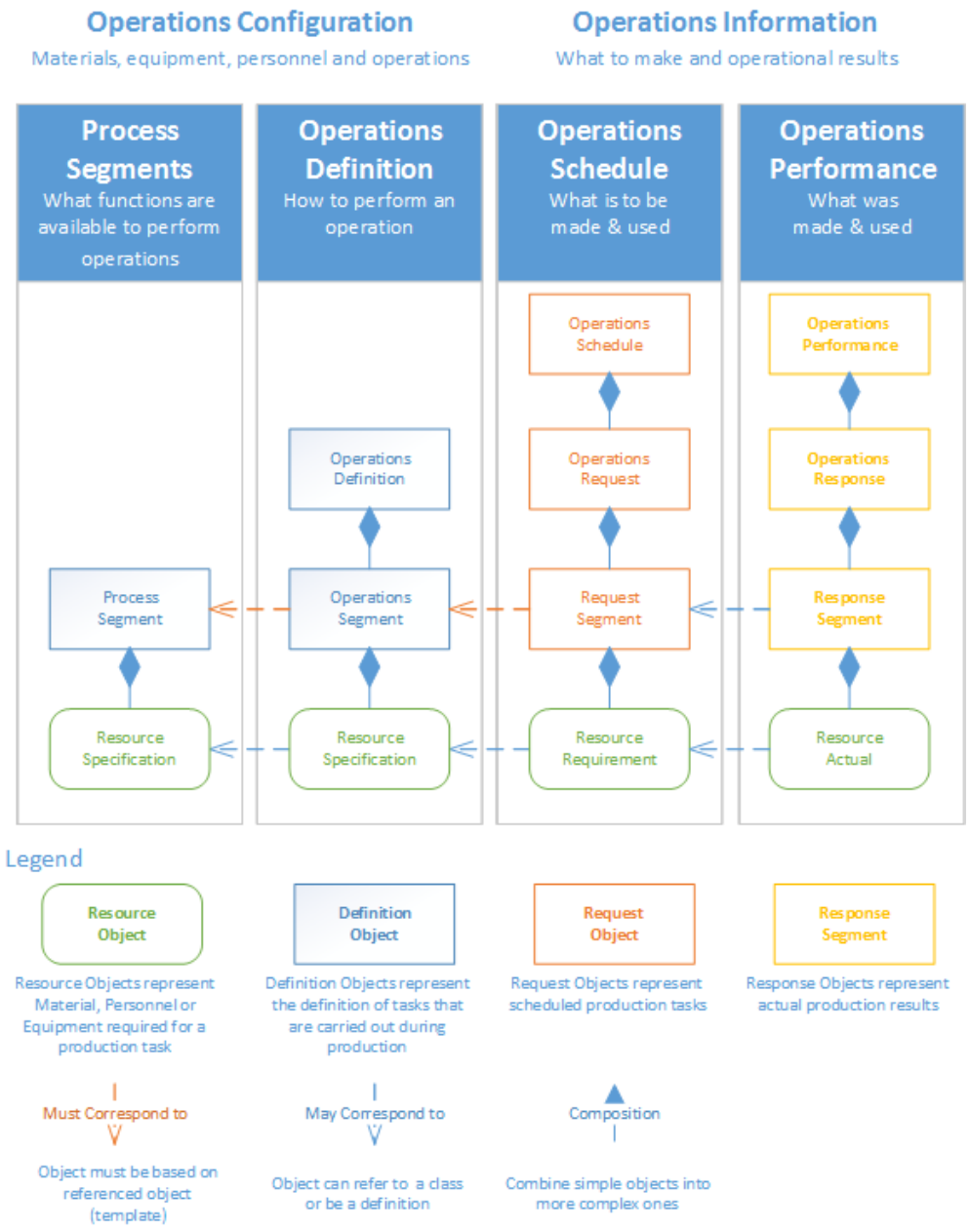
Sample UUID value: 5253ccae-47b4-4dc2-954f-900ffa8636eb

Mentally, it is hard to keep track of the full UUID value, so usually the last 3 or 4 digits will be unique and is easier to keep track in ones head.

9.6.2 MES Objects

At the core of the MES modules, objects are used extensively. Some are based on the ISA-95 standard and others are in addition to the ISA-95 standard. Collectively, we refer to all these objects as MES Objects. An object can represent material, equipment, task, etc. There are some properties, events and methods that are common across all objects. In addition to the common properties, events and methods, there are additional properties, events and methods based on the type of MES Object.

Today, the MES Objects are used by the Track & Trace and OEE 2.0 modules. In future releases, all other MES modules will also be based on the MES Objects.

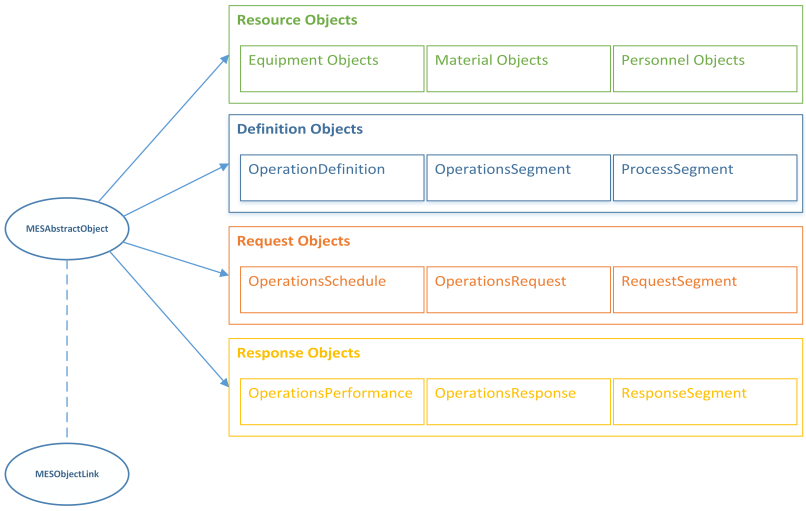


ISA-95 Object Model Inter-Relationships

The MES Objects are divided into four categories, **Resource**, **Definition**, **Request** and **Response**. These objects are all derived from the AbstractMESObject , inheriting its methods and properties as well as adding their own.

- **Resource** Objects represent equipment, material or personnel items that are required for production tasks.
- **Definition** Objects represent the definition of tasks that are carried out during production.
- **Request** objects represent scheduled production tasks

Response objects represent the actual production results.



There is a MES Object Link object . This is a light-weight object that acts like a reference to the full MES object, but only contains some general information. Consider using this when populating dropdowns or lists to reduce unneeded overhead.

Both the AbstractMESObject and the MES Object Link object s have a getMESObjectType() function that returns the type of an MES object . The MES Object Types object has the following helpful functions when working with MES Objects .

Object Functions

ISA-95 does not define object functions (or methods), but the MES modules extend these objects with functions to make common tasks easier. There are common functions that are available on all MES objects and then each MES object type may have additional functions. The additional functions are appropriate for the type of MES object.

An example of an object function that is common across all MES object is addCustomProperty(). An example of a object function that is specific for one type of MES object is setMaterial(). The setMaterial() function is only appropriate for a response segment type of MES object.

The Object model diagram in the Object Types section provides a quick view on which functions each object provides and which functions are inherited. Detailed information regarding the functions themselves can be found under the specific object in the Object Types section.

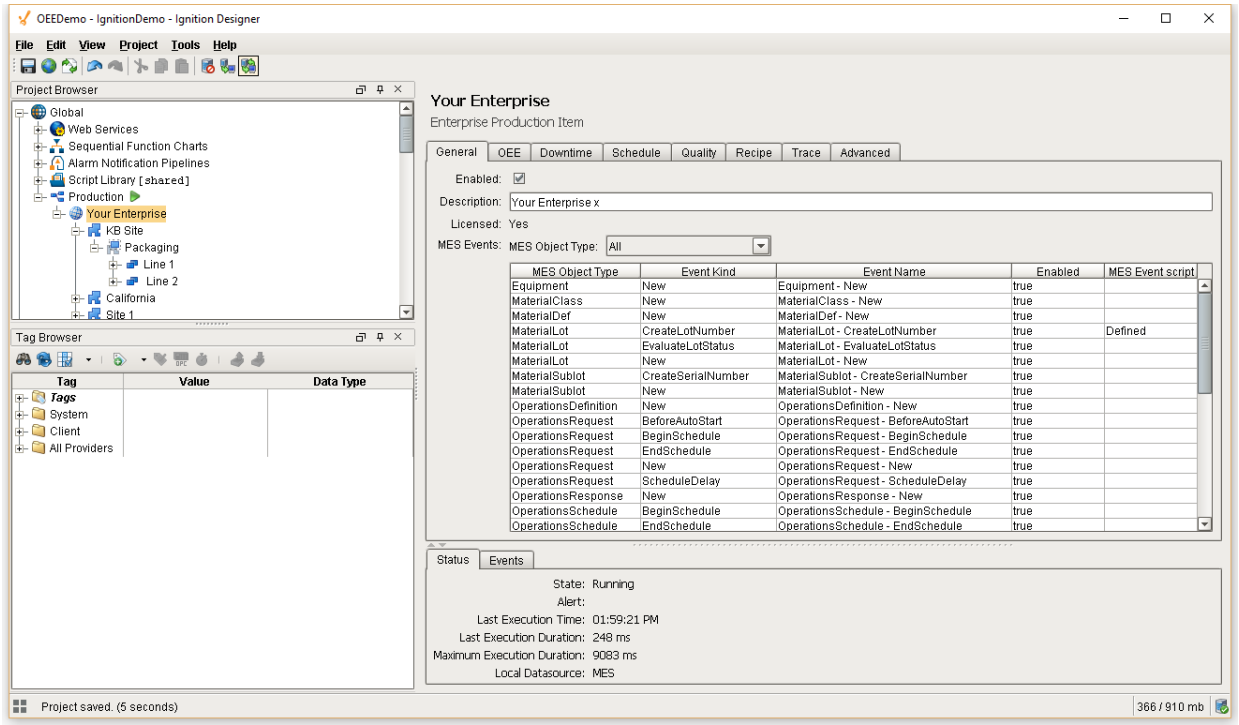
Scripting Functions

Scripting functions are provided through the system.mes namespace and can be found under Scripting Functions.

Object Events

Each of the different MES Object Types have events that are fired during the life cycle of the object. All MES object types support the **New** event that is run every time a new instance is created. This is great place to add custom properties for a given MES Object Type every time a new one is created. Depending on the type of MES object, there may be additional events that are fired. For example, the Material Lot MES object has an event that is run every time a new lot number is generated.

Events can be viewed and edited from the ProductionModel section of the Ignition Designer. Whenever an event is fired, an MES Script Event object is passed to the event. You can use this object to get information about the event and the object.



Event Types

There are two types of MES Object events, **System** events and **Custom** events.

System events are provided by default and cannot be deleted. A listing of all System Events is provided below. **Custom** Events can be created by right-clicking in the MES Events table and selecting 'New'.

For both System and custom events, custom scripts can be added to alter what happens when these event are triggered. This is also done in the ProductionModel section of the Ignition Designer.

List of System Events

MES Object Type	Event	Description
Equipment	New	The event is fired when a new instance of this MES object is created
MaterialClass	New	The event is fired when a new instance of this MES object is created
MaterialDef	New	The event is fired when a new instance of this MES object is created
MaterialLot	New	The event is fired when a new instance of this MES object is created
MaterialLot	CreateLotNumber	The event is fired when a Material Lot auto creates a new lot number. This event can be used to generate a uniquely patterned or sequenced lot number
MaterialLot	EvaluateLotStatus	The event is fired when a Material Lot is being finalized in a segment operation with a quantity and/or status change
MaterialSubLot	New	The event is fired when a new instance of this MES object is created
MaterialSubLot	CreateSerialNumber	The event is fired when a Material Sub Lot auto creates a new serial number. This event can be used to generate a uniquely patterned or sequenced serial number
OperationsDefinition	New	The event is fired when a new instance of this MES object is created
OperationsRequest	New	The event is fired when a new instance of this MES object is created

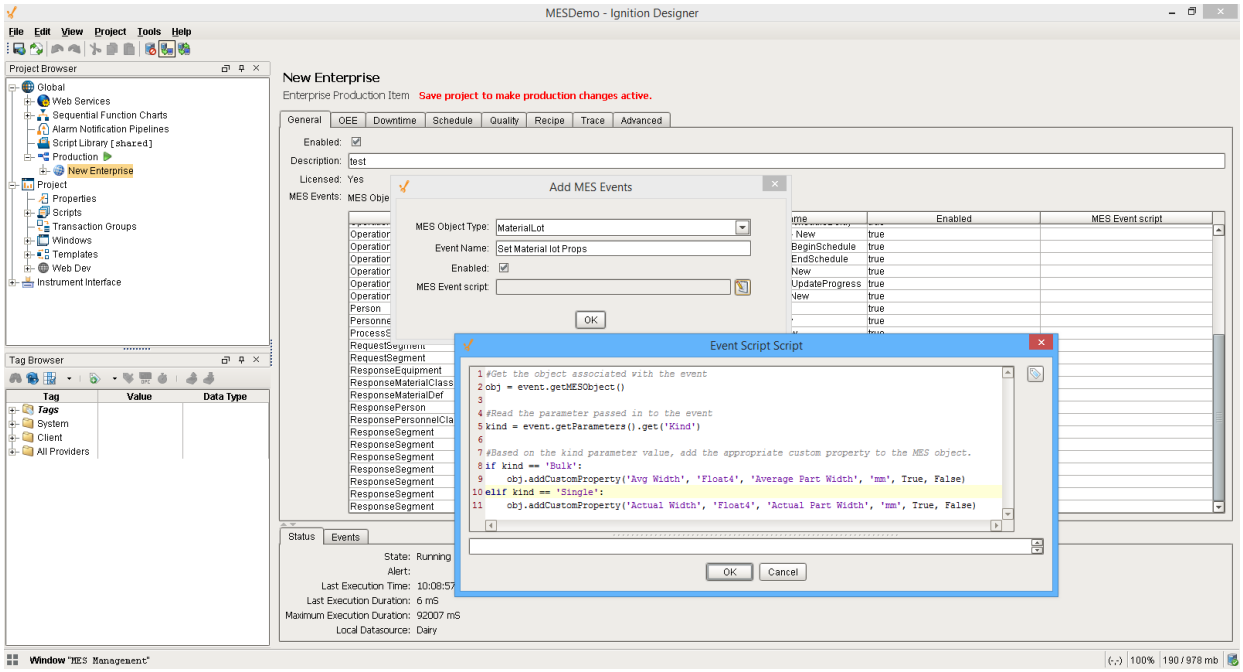
MES Object Type	Event	Description
OperationsRequest	BeforeAutoStart	The event is fired before the automatic start of a scheduled Operations Request
OperationsRequest	BeginSchedule	The event is fired before the requested operation is scheduled
OperationsRequest	EndSchedule	The event is fired after the requested operation has been scheduled
OperationsRequest	ScheduleDelay	The event is fired to give notification that an operation is delayed
OperationsResponse	New	The event is fired when a new instance of this MES object is created
OperationsSchedule	New	The event is fired when a new instance of this MES object is created
OperationsSchedule	BeginSchedule	The event is fired when an Operations Schedule has begun
OperationsSchedule	EndSchedule	The event is fired when an Operations Schedule has ended
OperationsSchedule	UpdateProgress	The event is fired during the update progress of an Operations Schedule. **Update Event** and **Update Event Interval** properties of the MES Object define when this event fires
Person	New	The event is fired when a new instance of this MES object is created
PersonnelClass	New	The event is fired when a new instance of this MES object is created
ProcessSegment	New	

MES Object Type	Event	Description
		The event is fired when a new instance of this MES object is created
RequestSegment	New	The event is fired when a new instance of this MES object is created
RequestSegment	Schedule	The Schedule Request Segment event is fired when a Request Segment is begin scheduled. This script must determine the begin date time and end date time of the request segment
ResponseEquipment	New	The event is fired when a new instance of this MES object is created
ResponseMaterialClass	New	The event is fired when a new instance of this MES object is created
ResponsematerialDef	New	The event is fired when a new instance of this MES object is created
ResponsePerson	New	The event is fired when a new instance of this MES object is created
ResponsePersonnelClass	New	The event is fired when a new instance of this MES object is created
ResponseSegment	New	The event is fired when a new instance of this MES object is created
ResponseSegment	BeforeAutoBegin	The event is fired before the automatic begin of a Response Segment
ResponseSegment	BeforeAutoEnd	The event is fired before the automatic end of a Response Segment
ResponseSegment	BeginTrace	

MES Object Type	Event	Description
		The event is fired when a trace of the Response Segment is started
ResponseSegment	EndTrace	The event is fired when a trace of the Response Segment is ended
ResponseSegment	SetRecipe	The event is fired when a recipe is set on the Response Segment
ResponseSegment	UpdateProgress	The event is fired during the update progress of a Response Segment. **Update Event** and **Update Event Interval** properties of the MES Object define when this event fires

Adding Custom Scripts

Custom scripts can be added to change the standard behavior of system events and add functionality when custom events are triggered. This is also done in the ProductionModel section of the Ignition Designer by right-clicking on the event and selecting 'Edit'.



Customizing a New Event

The **New** system event runs every time a new MES object is created. It can be used to add custom properties to the newly created object or to perform other tasks.

```
Add Custom Property to New MES Object Example

#Get the object associated with the event
obj = event.getMESObject()

#Read the parameter passed in to the event
kind = event.getParameters().get('Kind')

#Based on the kind parameter value, add the appropriate custom
property to the MES object.
if kind == 'Bulk':
    obj.addCustomProperty('Avg Width', 'Float4', 'Average Part
Width', 'mm', True, False)
elif kind == 'Single':
    obj.addCustomProperty('Actual Width', 'Float4', 'Actual Part
Width', 'mm', True, False)
```

If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line:

```
Add Custom Property and run defaultHandler

#Add custom property when a new instance of a MaterialDef object
is created.
obj = event.getMESObject()
obj.addCustomProperty('Width', 'Int4', 'Part Width', 'mm', True, F
alse)
event.runDefaultHandler()
```

Adding Custom Event

Custom Events can be created by right-clicking in the MES Events table in the ProductionModel section of the Ignition Designer and selecting 'New'.

The code block below shows how to generate a Custom event using the system.mes.executeMESEvent function.

```
Executing a Custom MES Event

#Load an MES object. In this case it is a MaterialLot object.
```

```
mesObject = system.mes.loadMESObject('VIN 3344', 'MaterialLot')

if mesObject != None:
    #Create parameters to send to the custom event.
    #Not required if no values are being passed to the custom
event
    params = system.mes.object.parameters.create()    #Creates an
MESObjectEventParameters object
    params.put('Kind', 'Dressing')
    #Add as many parameters as needed

    #Execute the custom event
    system.mes.executeMESEvent(mesObject, 'Set Material Lot Props'
, params)
```

References

MES Script Event

MES Object Event Parameters

system.mes.executeMESEvent

Object Properties

Properties are broken into three different categories. The first are core properties and follow closely to what the ISA-95 standard defines each object should support. The ISA-95 standard refers to these as attributes, but the Track and Trace module uses the term "core properties" to keep consistent with terminology already used in Ignition.

Next are custom properties, and the ISA-95 standards simply refers to these as properties. Again, the term is consistent with the existing "custom properties" term already being used in Ignition.

And last, there are complex properties. Complex properties are not defined explicitly in the ISA-95 standard, but are inferred when defining resources needed for an operation and other aggregate lists.

Complex Properties

Complex properties are predefined and vary based on the type of MES object. In general, they are named complex properties because they have a number of members and there can be multiple instances for a given complex property type. A Process Segment to mix dressing will have multiple material references. Each one will be an instance of a Material complex property. Even more complex is the Lot complex property used by the Response Segment for mixing

dressing, because there will be multiple lots for each input material each with their own complex property base name. Then, if an input material lot is used up during production and switched over to another input lot, then there will be multiple instances for the same input lot reference.

An example will help clarify this concept. If we are mixing dressing which requires vinegar that is coming from Lot 123 in vinegar tank 1, then there will be an instance of a Lot complex property for it. This complex property has a name to reference it by, and in this example we will call it "Ingredient Vinegar". Then during production of mixing dressing, we run out of vinegar in tank 1 and we switch over to vinegar tank 2. Vinegar tank 2 contains Lot 234 so there will be another Lot complex property for it with the same name "Ingredient Vinegar". Now there are two Lot complex properties that are referenced by the same name "Ingredient Vinegar". Behind the scenes, the names are modified with a post fix. As a result the Lot complex property for Lot 123 will be named "Ingredient Vinegar-1" and the one for Lot 234 will be named "Ingredient Vinegar-2". This is referred to as extended naming. Each will hold complete details of when the lot start and when it finished, quantity, etc.

Refer to the Process Segment, Operations Definition, Operations Segment, Operations Response. and Response Segment for more details about the different types of complex properties.

Add Complex Property

Code Snippet

```
#Create and add complex property using script
#Create a new process segment
seg = system.mes.createMESObject('ProcessSegment')

#Create a new material reference complex property
materialProp = mesObject.createComplexProperty('Material', 'Vinega
r')

#Add the new material reference complex property to the new
segment
seg.addComplexProperty(materialProp)

#Do more stuff...

#Save the new object
system.mes.saveMESObject(seg)
```

Code Snippet

```
#Read all complex properties and print them
#Read process segment MES object named Mix Dressing
mesObject = system.mes.loadMESObject('Mix Dressing', 'ProcessSegme
nt')

#Get all available complex property types and cycle through them
list = mesObject.getComplexPropertyTypeNames()
for i in range(list.size()):
    complexPropType = list.get(i)

    #Get the number of entries for the current complex property
type and cycle through them and print the name
    cnt = mesObject.getComplexPropertyCount(complexPropType)
    for j in range(cnt):
        complexProp = mesObject.getComplexProperty
(complexPropType, j)
        print complexProp.getName()
```

Core Properties

The base AbstractMESObject object provides common core properties that are passed on to each of specific MES objects types. Each specific MES object can have additional core properties.

Core Properties common to all MES Objects

Setting Name	Type	Description
name	read only	This is the name of the MES object. This name is used when referencing the object. It must be a unique name meaning that no other MES object of it's type can have the same name.
UUID	read only	This will contain the Universally Unique Identifier for each instance of a MES object.
enabled	write only	This property will be set to true when the MES object is active and usable. When MES objects are deleted they are still retained in the database and the Enabled setting is set to false. This is done to maintain past traceability information.
description	read-only	An optional settings to give more details for a MES Object.

The values of the core properties are accessed using the getPropertyValue() and setPropertyValue() functions of the AbstractMESObject object.

Examples

```
Read Object UUID Snippet

#Get MES object UUID example:
mesObject = system.mes.loadMESObject("Vinegar", "MaterialClass")
print mesObject.getPropertyValue('UUID')
```

```
Set Object Name Snippet

#Set MES object name example:
mesObject = system.mes.loadMESObject("Vinegar", "MaterialClass")
mesObject.setPropertyValue('Name', 'MyNewObjectName')
```

Custom Properties

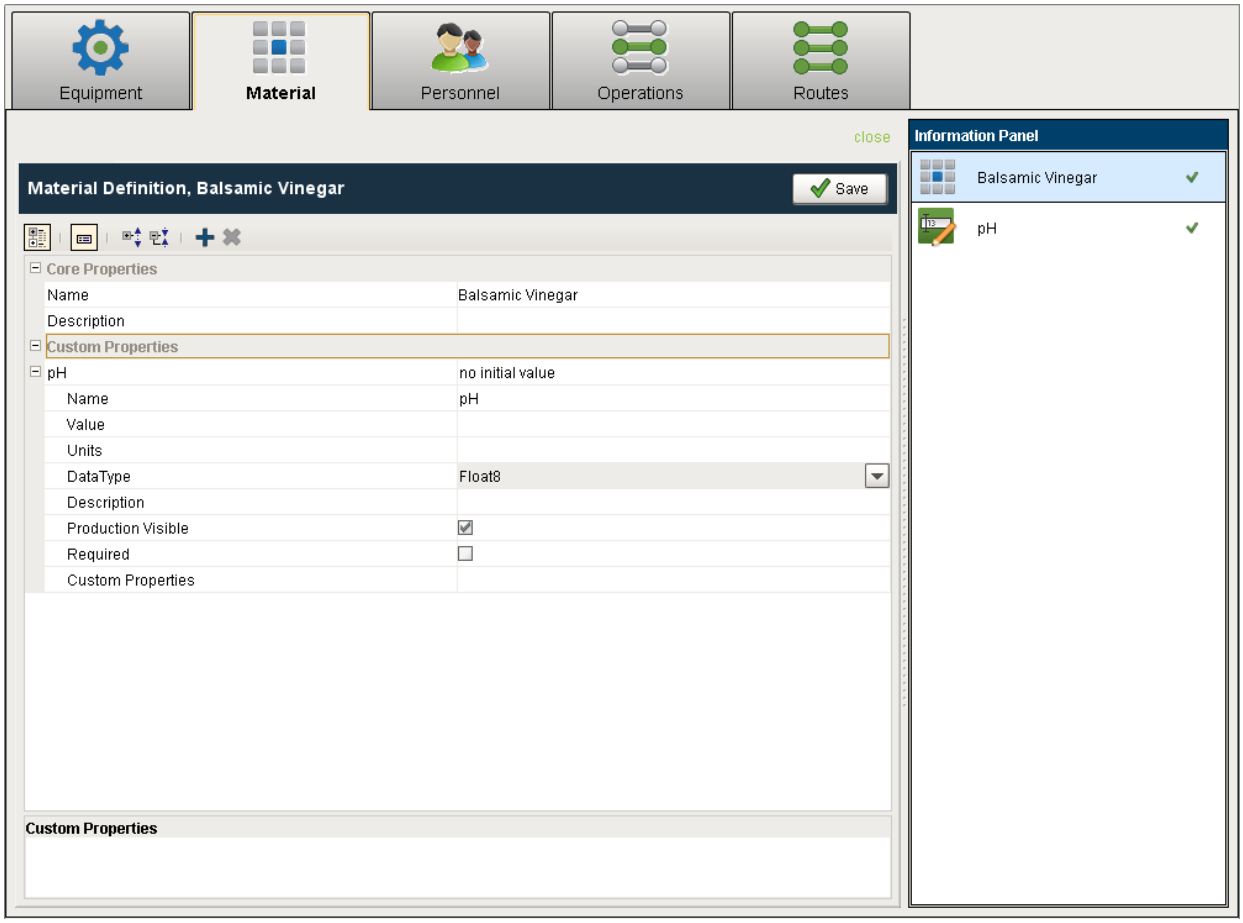
Custom properties are added to any MES object by using the MES object editor component or by using script functions.

Custom properties can be nested, meaning a custom property can be added to a custom property. This allows defining a structure to custom properties where Width, Height and Depth custom properties can be add beneath Dimension custom property.

Custom properties have a production visible option that will show the property in the MES Property Value Editor component. This provides a method to keep custom properties hidden and not show then to operators or other end users. If the custom property is visible to the end users, the required options will make sure a value is entered before ending an operation.

Adding / editing custom property using the MES Object Editor

The MES Object Editor component has built-in support to add custom properties to any of the MES object types. The only MES object types that custom properties cannot be added using the MES Object Editor Component are ones not configured using the MES Object Editor. These include MES objects like MaterialLot, MaterialSublot, OperationsResponse and ResponseSegment, because they are only used during production and not used to define resources or operations.



Code Snippet

```
#Read custom property from MES object.

#Load the material class object named Vinegar.
mesObject = system.mes.loadMESObject("Vinegar", "MaterialClass")

#Print value of custom property pH
print mesObject.getPropertyValue('pH')
```

Code Snippet

```
#Add new custom property to MES object.

#Load the material class object named Vinegar.
mesObject = system.mes.loadMESObject('Vinegar', 'MaterialClass')

#Add a new custom property named pH.
mesObject.addCustomProperty('pH', 'Float8')
```

```
#Set the value of pH to 5.1
mesObject.setPropertyValue('pH', 5.1)
```

Code Snippet

```
#Add new custom property to MES object.

#Load the material class object named Mounting Plate.
mesObject = system.mes.loadMESObject('Mounting Plate', 'MaterialDe
f')

#Add a new custom property named Width.
mesObject.addCustomProperty('Width', 'Float8', 'Width of mounting
plate', 'mm', True, True)
```

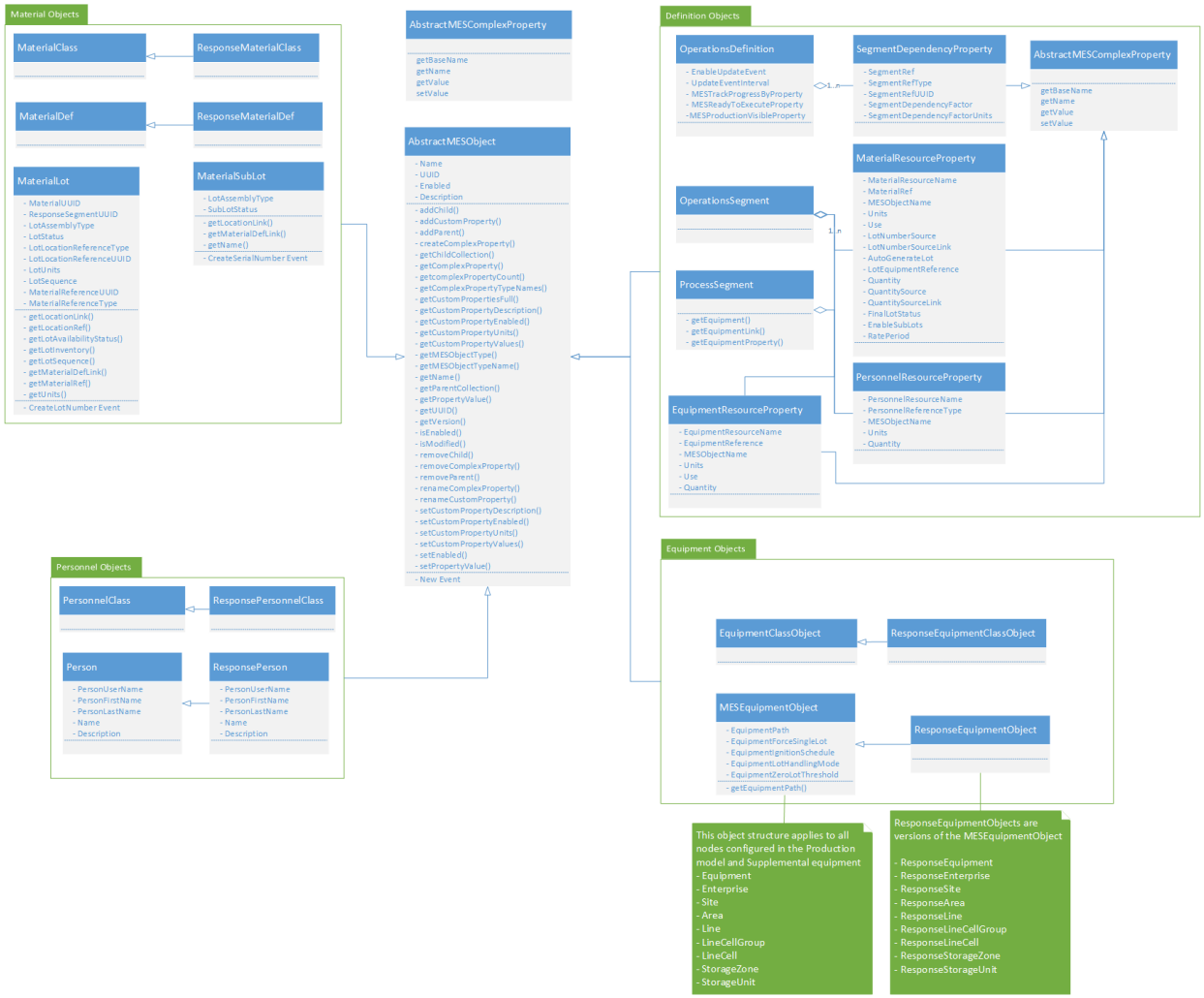
Info

See the custom property section of the AbstractMESObject reference for all of the custom property functions.

Object Types

There are many different types of MES objects in the Sepasoft MES Module, all of which are inherited from the AbstractMESObject . Many of the scripting functions and properties refer to the common AbstractMESObject . The image below shows the MES Objects properties, methods and events. As all objects inherit from the AbstractMESObject , they all will have the **Name** property and the **addCustomProperty()** method.

As an example, the Equipment Object has an **EquipmentPath** property and a **getEquipmentPath()** method in addition to those properties and methods inherited from the AbstractMESObject .



MESObjectTypes

There is a MESObjectTypes object has some helpful functions when working with MES Objects . Both the AbstractMESObject and the MES Object Link object s have a getMESObjectType() function that returns the type of an MES object .

The code below shows how to determine the specific MES object type using the getMESObjectType() method on the object.

Get MES Object Type

```
#This code snippet will print the names of MES object types.
filter = system.mes.object.filter.createFilter()
filter.setMESObjectNamePattern('Receive Turkeys')
list = system.mes.searchMESObjects(filter)
for ndx in range(list.size()):
    mesObject = list.get(ndx)
    print mesObject.getMESObjectType().getDisplayName()
```

AbstractMESComplexProperty

An AbstractMESComplexProperty object is used as a base object for Material Resource Property, Personnel Resource Property, Equipment Resource Property (which are detailed in Process Segment reference), the Segment Dependency Property (which is detailed in the Operations Definition reference) and the Lot Reference Property (which is detailed in the Response Segment reference).

The complex property script functions return an AbstractMESComplexProperty object. This is done because duplicate functions would have to be implemented for Material Resource Property, Personnel Resource Property, etc. reference type.

Code Snippet

```
#Read Complex Property Value Example:
mesObject = system.mes.loadMESObject('Unload Vinegar', 'ProcessSeg
ment')
matRef = mesObject.getComplexProperty('Material', 'Raw Material')
print matRef.getValue('MaterialUse')
```

Output

```
Out
```

Complex properties have various values of various data types. For the specific value names see:

- Process Segment for details about Material Resource Property, Personnel Resource Property and Equipment Resource Property.
- Operations Definition for details about Segment Dependency.
- Operations Response for details about Lot.

getBaseName

Complex properties that use extended naming have an associated base name and a number following it. This is used with Lot Reference Property that the Response Segment uses. Each time a lot reference is created it is named the base name with an extension added to it.

For example:

If balsamic vinegar is being unloaded and the tank it is being stored in fills up before the truck is fully unloaded. When it is switched to another storage tank a new lot reference is created. If the name of the material reference in the Process Segment is named Vinegar, then the first lot reference is named Vinegar-1 for the first storage tank and the second is Vinegar-2 for the second storage tank.

Description

Get the base name for the complex property.

Syntax

getBaseName()

- Parameters

None

- Returns

String baseName - The base name for the complex property.

- Scope

All

Code Examples

Code Snippet

```
#This code snippet prints the base name for the specific
property.
seg = system.mes.createMESObject('ProcessSegment')
#Create a new material reference complex property
materialProp = seg.createComplexProperty('Material', 'Vinegar')
print materialProp.getBaseName()
```

Output

```
Vinegar
```

getName

Complex properties that use extended naming have an associated base name and a number following it. This is used with Lot Reference Property that the Response Segment uses. Each time a lot reference is created it is named the base name with an extension added to it.

For example:

If balsamic vinegar is being unloaded and the tank it is being stored in fills up before the truck is fully unloaded. When it is switched to another storage tank a new lot reference is created. If the name of the material reference in the Process Segment is named Vinegar, then the first lot reference is named Vinegar-1 for the first storage tank and the second is Vinegar-2 for the second storage tank.

Description

Get the name for the complex property.

Syntax

getName()

- Parameters

None

- Returns

String - The name for the complex property.

- Scope

All

Code Examples

```
Code Snippet
```

```
seg = system.mes.createSegment('Receive Steel', '[global]
\Enterprise\Site\Area\Unload Station 1', True)
seg.setMaterial('Steel In', '84000', '[global]
\Enterprise\Site\Receiving\Steel\QC Holding', 'Lot 84000-1', 10
0.0)
cnt = seg.getComplexPropertyCount('ResponseMaterial')
for ndx in range(0, cnt):
 prop = seg.getComplexProperty('ResponseMaterial', ndx)
 print prop.getName()
seg.begin()
seg = system.mes.getActiveSegment('Enterprise\Site\Area\Unload
Station 1', 'Receive Steel')
seg.setMaterial('Steel In', '84001', '[global]
\Enterprise\Site\Receiving\Steel\QC Holding', 'Lot 84001-1', 10
0.0)
seg.update()
seg = system.mes.getActiveSegment('Enterprise\Site\Area\Unload
Station 1', 'Receive Steel')
cnt = seg.getComplexPropertyCount('ResponseMaterial')
for ndx in range(0, cnt):
 prop = seg.getComplexProperty('ResponseMaterial', ndx)
 print prop.getName()
```

Output

```
Steel In-1
Steel In-2
```

getValue

Description

Returns the value for the specific property.

Syntax

getValue(valueName)

- Parameters

String valueName - The name of the value to return.

- Returns

String - The value as an object.

- Scope

All

Code Examples

Code Snippet

```
#This code snippet prints the value for the specific property.
seg = system.mes.createMESObject('ProcessSegment')
#Create a new material reference complex property
materialProp = seg.createComplexProperty('Material', 'Vinegar')
print materialProp.getValue('MaterialProductionSelectable')
```

Output

```
True
```

setValue

Description

Set the value for the specified value name.

Syntax

setValue(valueName, value)

- Parameters

String valueName - The name of the value to set.

Serializable value - The new value. A string value is recommended.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Example
seg = system.mes.createMESObject('ProcessSegment')
#Create a new material reference complex property
materialProp = seg.createComplexProperty('Material', 'Vinegar')
materialProp.setValue('MaterialRatePeriod', 'Sec')
materialProp.setValue('MaterialRate', '74')
#Just to make sure the value is reset.
print materialProp.getValue('MaterialRatePeriod')

##Save the segment to manifest the changes.
system.mes.saveMESObject(seg)
```

Output

```
Sec
```

Trigger Operation Begin Property

This is an extension to the Abstract MES Complex property and it creates a trigger in the operation segment.

Properties:

getMode()

Description

Returns the mode of the complex property.

Syntax

getMode()

- Parameters

None

- Returns

String mode - The mode of this property.

- Scope

All

getPrecedingRef()

Description

Gets the preceding reference to this complex property.

Syntax

getPrecedingRef()

- Parameters

None

- Returns

MESObjectLink - The MES object link to the preceding reference to this complex property.

- Scope

All

getPrecedingRefProperty()

Description

Gets the preceding reference property.

Syntax

getPrecedingRefProperty()

- Parameters

None

- Returns

MESTriggerOperationBeginPrecedingRefProperty - The preceding reference property.

- Scope

All

getPrecedingRefType()

Description

Gets the preceding reference type.

Syntax

getPrecedingRefType()

- Parameters

None

- Returns

String precedingRefType - Type of the preceding reference to this complex property.

- Scope

All

Code Examples

```
Code Snippet
```

getPrecedingRefUUID()

Description

Gets the preceding reference uuid.

Syntax

getPrecedingRefUUID()

- Parameters

None

- Returns

String precedingRefUUID - The uuid corresponding to the preceding reference.

- Scope

All

Code Examples

```
Code Snippet
```

isAuto()

Description

Checks whether the auto execute property is set to True.

Syntax

isAuto()

- Parameters

None

- Returns

boolean - True if the Begin trigger auto execute property is true and False otherwise.

- Scope

All

Code Examples

```
Code Snippet
```

isDefault()

Description

Checks whether the default trigger property is set to True.

Syntax

isDefault()

- Parameters

None

- Returns

boolean - True if the default trigger property is true and False otherwise.

- Scope

All

Code Examples

```
Code Snippet
```

isModePrimary()

Description

Checks whether the mode of the property is primary.

Syntax

isModePrimary()

- Parameters

None

- Returns

boolean - True if the property mode is primary and False otherwise.

- Scope

All

Code Examples

Code Snippet

requiresReference()

Description

Checks whether the property is in reference mode.

Syntax

requiresReference()

- Parameters

None

- Returns

boolean - True if the property is in reference mode and False otherwise.

- Scope

All

Code Examples

Code Snippet

setAuto(auto)

Description

Sets the auto execute property to a boolean.

Syntax

setAuto(auto)

- Parameters

boolean - True to set the Begin trigger auto execute property to true and False otherwise.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet
```

setDefault(value)

Description

Sets a default value to the complex property.

Syntax

setDefault(value)

- Parameters

Boolean value - True to set the default value to True and False otherwise.

- Returns

Nothing

- Scope

All

setMode(mode)

Description

Sets the mode of the complex property.

Syntax

setMode(mode)

- Parameters

String mode - The mode to set for this property.

- Returns

Nothing

- Scope

All

setPrecedingRef(mesObjectLink)

Description

Sets the preceding reference of this complex property.

Syntax

setPrecedingRef(mesObjectLink)

- Parameters

MESObjectlink mesObjectLink - The MES object link to the preceding reference to set for.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet
```

setPrecedingRefType(precedingRefType)

Description

Sets the preceding reference type.

Syntax

setPrecedingRefType(precedingRefType)

- Parameters

String precedingRefType - The type to set for the preceding reference.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet
```

setPrecedingRefUUID(precedingRefUUID)

Description

Sets the preceding reference uuid.

Syntax

setPrecedingRefUUID(precedingRefUUID)

- Parameters

String precedingRefUUID - The uuid to set for the preceding reference.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet
```

Trigger Segment Begin Property

This is an extension to the Abstract MES Complex property and it creates a trigger when the segment begins.

Properties:

getMESPropertyID()

Description

Gets the id of the MES property.

Syntax

getMESPropertyID()

- Parameters

None

- Returns

MESPropertyID - The identifier corresponding to this property.

- Scope

All

getPrecedingRef()

Description

Gets the preceding reference to this property.

Syntax

getPrecedingRef()

- Parameters

None

- Returns

MESObjectLink - The reference of the preceding MES object.

- Scope

All

setPrecedingRef(mesObjectLink)

Description

Sets the preceding reference to this property.

Syntax

setPrecedingRef(mesObjectLink)

- Parameters

MES Object Link mesObjectLink - The reference of the preceding MES object to set for.

- Returns

Nothing

- Scope

All

getPrecedingRefUUID()

Description

Gets the uuid of the MES object that precedes this property.

Syntax

getPrecedingRefUUID()

- Parameters

None

- Returns

String uuid - The uuid corresponding to the object that precedes this property.

- Scope

All

setPrecedingRefUUID(precedingRefUUID)

Description

Sets the uuid of the MES object that precedes this property.

Syntax

setPrecedingRefUUID(precedingRefUUID)

- Parameters

String precedingRefUUID - The uuid to set for the preceding MES object.

- Returns

Nothing

- Scope

All

getPrecedingRefProperty()

Description

Gets the preceding reference property.

Syntax

getPrecedingRefProperty()

- Parameters

None

- Returns

MESTriggerOperationBeginPrecedingRefProperty - The preceding reference property.

- Scope

All

AbstractMESObject

There are many different types of MES objects in the Sepasoft MES system. All of them are inherited from the AbstractMESObject. Many of the scripting functions and properties refer to the commonAbstractMESObject objects. This page details the properties, functions and events that are common to all objects that are inherited from the AbstractMESObject.

Core Properties common to all MES Objects

Setting Name	Type	Description
name	read-only	This is the name of the MES object. This name is used when referencing the object. It must be a unique name meaning that no other MES object of it's type can have the same name.
UUID	read-only	This will contain the Universally Unique Identifier for each instance of a MES object.
enabled	write-only	This property will be set to true when the MES object is active and usable. When MES objects are deleted they are still retained in the database and the Enabled setting is set to false. This is done to maintain past traceability information.
description	read-only	An optional settings to give more details for a MES Object.

Events

'New' Event

This event is run every time a new MES object is created. It can be used to add custom properties or to perform other tasks.

If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line:

Code Snippet

```
#Add custom property when a new instance of a MaterialDef object
is created.
obj = event.getMESObject()
obj.addCustomProperty('Width', 'Int4', 'Part Width', 'mm', True, F
alse)
event.runDefaultHandler()
```

Script Functions

The script functions listed below are available for all MES objects. They are used to simplify and reduce the number of lines of script for common tasks. An example is adding children, adding custom properties, changing property values, etc.

All of these script functions require an instance of a MES object. There are a number of methods to get an instance of an MES object and the code snippets below show just a couple of them.

Code Snippet

```
#Get the MES object for a given name and MES object type
obj = system.mes.loadMESObject('Balsamic Vinegar', 'MaterialDef')
```

Code Snippet

```
#If a link was returned from another script function, then this
will return the full MES object instance
obj = objLink.getMESObject()
```

addChild

Description

MES objects can have children and parents. Depending on the type of MES object determines the types of children or parents that can be added. For example, a material class (MESMaterialClass) object can have material definitions (MESMaterialDef) as children, but material definitions objects cannot have material class objects as children.

Note

When a child is added to an MES object, then a parent reference will be added to the child behind the scenes. Likewise, when a parent is added to a child, a child reference will be added to the parent behind the scenes. This insures if integrity of relationships are maintained.

Method Options

addChild(mesObject)

Description

Add a MES object as a child to another MES object

Syntax

addChild(mesObject)

- Parameters

AbstractMESObject mesObject - An instance to an MES object. An AbstractMESObject object is just the generic form of an MES object when the specific type is unknown.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Get the Screws material class object
matCls = system.mes.loadMESObject('Screws', 'MaterialClass')
#Get the 10-32 NC Screw material definition object
matDef = system.mes.loadMESObject('10-32 NC Screw', 'MaterialDe
f')
#Add the 10-32 NC Screw material definition object as a child
to the Screws material class object
matCls.addChild(matDef)
#Save the changes
system.mes.saveMESObject(matCls)
```

addChild(mesObjectLink)

Description

Add a MES object link as a child to another MES object.

Syntax

addChild(mesObject)

- Parameters

MES Object Link mesObjectLink - A link to an MES object. A MES Object Link holds identification information to a MES object. This is very efficient when displaying MES objects in a list and all the MES object details are not needed.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Get the Storage Tanks equipment class object
eqCls = system.mes.loadMESObject('Storage Tanks', 'EquipmentCla
ss')
#Get the Tank 1A equipment object from the equipment path
eq = system.mes.getMESObjectLinkByEquipmentPath('[global]
\Dressings Inc\California\Raw Materials\Tank Farm\Tank 1A')
#Add the 10-32 NC Screw material definition object as a child
to the Screws material class object
eqCls.addChild(eq)
#Save the changes
system.mes.saveMESObject(eqCls)
```

addCustomProperty

Description

Custom properties can be added to MES objects or even complex properties (such as a material, equipment or personnel reference) of an MES object. The addCustomProperty method can be used to add a custom property directly to a MES object by not specifying a path to the parent property. To add a custom property as a child of an existing custom property or to a complex property, the parent path is used to specify the parent.

Each level of the path is separated with a period. This allows for names to be duplicated provided the path is unique. For example, it is possible for a MES object to have custom properties "Dimension 1.Wdith" and "Dimension 2.Width"

Method Options

addCustomProperty(name, dataTypeName, description, units, productionVisible, required, value)

Syntax

addCustomProperty(name, dataTypeName, description, units, productionVisible, required, value)

- Parameters

String name - The name of the custom property to add.

String dataTypeName - The name of the Ignition data type to make the new custom property.

String description - The description of the custom property.

String units - The units of the new custom property. This is just for reference.

Boolean productionVisible - **The default is false.** If True, show the custom property in various components. If False, it will be hidden and can be used to store values behind the scenes.

Boolean required - If True, a value must be assigned to the custom property before and segment is ended.

String value - The value to assign to the custom property after it has been added.

- Returns

Nothing

- Scope

All

addCustomProperty(name, dataType, description, units, productionVisible, required)

Syntax

addCustomProperty(name, dataTypeName, description, units, productionVisible, required)

- Parameters

String name - The name of the custom property to add.

String dataTypeName - The name of the Ignition data type to make the new custom property.

String description - The description of the custom property.

String units - The units of the new custom property. This is just for reference.

Boolean productionVisible - **The default is false.** If True, show the custom property in various components. If False, it will be hidden and can be used to store values behind the scenes.

Boolean required - If True, a value must be assigned to the custom property before and segment is ended.

- Returns

Nothing

- Scope

All

addCustomProperty(name, dataType)

Syntax

addCustomProperty(name, dataTypeName)

- Parameters

String name - The name of the custom property to add.

String dataTypeName - The name of the Ignition data type to make the new custom property.

- Returns

Nothing

- Scope

All

addCustomProperty(parentPath, name, dataType, description, units, productionVisible, required, value)

Syntax

addCustomProperty(parentPath, name, dataTypeName, description, units, productionVisible, required, value)

- Parameters

String parentPath - The path of the parent to add the custom property to.

String name - The name of the custom property to add.

String dataTypeName - The name of the Ignition data type to make the new custom property.

String description - The description of the custom property.

String units - The units of the new custom property. This is just for reference.

Boolean productionVisible - **The default is false.** If True, show the custom property in various components. If False, it will be hidden and can be used to store values behind the scenes.

Boolean required - If True, a value must be assigned to the custom property before and segment is ended.

String value - The value to assign to the custom property after it has been added.

- Returns

Nothing

- Scope

All

addCustomProperty(parentPath, name, dataType, description, units, productionVisible, required)

Syntax

addCustomProperty(parentPath, name, dataTypeName, description, units, productionVisible, required)

- Parameters

String parentPath - The path of the parent to add the custom property to.

String name - The name of the custom property to add.

String dataTypeName - The name of the Ignition data type to make the new custom property.

String description - The description of the custom property.

String units - The units of the new custom property. This is just for reference.

Boolean productionVisible - **The default is false.** If True, show the custom property in various components. If False, it will be hidden and can be used to store values behind the scenes.

Boolean required - If True, a value must be assigned to the custom property before and segment is ended.

- Returns

Nothing

- Scope

All

addCustomProperty(parentPath, name, dataType)

Syntax

addCustomProperty(parentPath, name, dataTypeName)

- Parameters

String parentPath - The path of the parent to add the custom property to.

String name - The name of the custom property to add.

String dataTypeName - The name of the Ignition data type to make the new custom property.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Load a MES object
obj = system.mes.loadMESObject('Box', 'MaterialDef')

#Add Type custom property directly to the MES object
obj.addCustomProperty('Type', 'String')

#Add Dimension custom property directly to the MES object
obj.addCustomProperty('Dimension', 'String', 'Dimension of box'
, '', True, False)

#Add Width custom property to the previous Dimension custom
property and assign a value
obj.addCustomProperty('Dimension', 'Width', 'Int4', 'Width of
box', 'in', True, False, '24')
```

```
#Add Height custom property to the previous Dimension custom
property and assign a value
obj.addCustomProperty('Dimension', 'Height', 'Int4', 'Height
of box', 'in', True, False, '12')

#Type custom property was never assigned a value so None is
returned
print obj.getPropertyValue('Type')

#Remember to save the MES object
system.mes.saveMESObject(obj)

#Width and Height custom properties were assigned values
print obj.getPropertyValue('Dimension.Width')
print obj.getPropertyValue('Dimension.Height')
```

Output

```
None
24
12
```

addParent

Description

MES objects can have children and parents. Depending on the type of MES object determines the types of children or parents that can be added. For example, a material class (MESMaterialClass) object can have material definitions (MESMaterialDef) as children, but material definitions objects cannot have material class objects as children.

Note

When a child is added to an MES object, then a parent reference will be added to the child behind the scenes. Likewise, when a parent is added to a child, a child reference will be added to the parent behind the scenes. This insures if integrity of relationships are maintained.

Method Options

addParent(mesObject)

Description

Add a MES object as a parent to another MES object

Syntax

addParent(mesObject)

- Parameters

AbstractMESObject mesObject - An instance to an MES object. An AbstractMESObject object is just the generic form of an MES object when the specific type is unknown.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Get the 10-32 NC Screw material definition object
matDef = system.mes.loadMESObject('10-32 NC Screw', 'MaterialDe
f')
#Get the Screws material class object
matCls = system.mes.loadMESObject('Screws', 'MaterialClass')
#Add the Screws material class object as a parent to the 10-32
NC Screw material definition object
matDef.addParent(matCls)
#Save the changes
system.mes.saveMESObject(matDef)
```

addParent(mesObjectLink)

Description

Add a MES object link as a parent to another MES object

Syntax

addParent(mesObject)

- Parameters

MES Object Link mesObjectLink - A link to an MES object. A MES Object Link holds identification information to a MES object. This is very efficient when displaying MES objects in a list and all the MES object details are not needed.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Get the Tank 1A equipment object from the equipment path
eq = system.mes.getMESObjectLinkByEquipmentPath('[global]
\Dressings Inc\California\Raw Materials\Tank Farm\Tank 1A')
#Get the Storage Tanks equipment class object
eqCls = system.mes.loadMESObject('Storage Tanks', 'EquipmentCla
ss')
#Add the Screws material class object to the 10-32 NC Screw
material definition object
eq.addChild(eqCls)
#Save the changes
system.mes.saveMESObject(eq)
```

createComplexProperty

Description

Create a complex property directly to an MES object by not specifying a path to the parent property.

Syntax

createComplexProperty(complexPropertyType , name)

- Parameters

String complexPropertyType - The type of complex property.

String name - The name of complex property.

- Returns

The complex property for the specified type.

- Scope

All

Code Examples

Code Snippet

```
od = system.mes.createMESObject('OperationsDefinition')

#Load the process segment to base the operations segments on
ps = system.mes.loadMESObject('Receive Material', 'ProcessSegme
nt')
depProp = od.createComplexProperty('SegmentDependency', ps.
getName())

#Derive a new operation segment from the process segment
os = system.mes.deriveMESObject(ps, 'OperationsSegment', True)
depProp.setSegmentRefType('OperationsSegment')
depProp.setSegmentRefUUID(os.getUUID())
print depProp
```

Output

```
{SegmentRef=Operations Segment, , SegmentRefUUID=cfabeb4a-0b2d-
4e43-91a8-1258d8b41e32, SegmentRefType=OperationsSegment,
SegmentDependencyType=null, SegmentDependencyFactor=null,
SegmentDependencyFactorUnits=null}
```

getAllCustomProperties

Description

Returns all the custom properties for the definition and any class object that the definition is a member of.

Syntax

getAllCustomProperties()

- Parameters

None

- Returns

List<MESCustomProperty> - The list of custom properties.

- Scope

All

Code Examples

Code Snippet

```
matDef = system.mes.loadMESObject('Cane Sugar', 'MaterialDef')
matDef.getAllCustomProperties()
```

getChildCollection

Description

Get the collection of children for an MES object.

Syntax

getChildCollection()

- Parameters

None

- Returns

MES Object Collection - Returns a collection object containing references to all children of the MES object.

- Scope

All

MESObjectCollection Details

A MESObjectCollection object is used by MES objects to hold parents and children. Normally, the parent and child script functions of the MES objects should be used, but this is provided as a reference to the MESObjectCollection object itself and provides some additional functionality.

Methods:

get(uuid)

Description

Returns the MES object link for the specified UUID. If the specified UUID does not exist, None will be returned.

Syntax

get(uuid)

- Parameters

None

- Returns

String mesObjectLink - The MES object link corresponding to the uuid.

Code Examples

Code Snippet

```
#Object link corresponding to the specified uuid is returned.
mesObject = system.mes.loadMESObject('Vinegar', 'MaterialClass'
)
if mesObject != None:
    childList = mesObject.getChildCollection()
    print childList.get('5acf3c9f-2789-44af-888f-fce08d9972a7')
```

Output

```
Red Wine Vinegar
```

getList()

Description

Returns a list of MES object links. Depending if getParentCollection() or getChildCollection() is called to get the MESObjectCollection object, it will contain MES object links that are parents or children.

Syntax

getList()

- Parameters

None

- Returns

String mesObjectLinkList - A list containing MES object links.

Code Examples

Code Snippet

```
#This example reads the child MES object links that belong to
the Vinegar Material Class.
mesObject = system.mes.loadMESObject('Vinegar', 'MaterialClass'
)
if mesObject != None:
    childList = mesObject.getChildCollection().getList()
    for child in childList:
        print child.getName()
```

Output

```
Balsamic Vinegar
Red Wine Vinegar
White Vinegar
```

isEmpty()

Description

Returns True if no MES object links exist in the collection.

Syntax

isEmpty()

- Parameters

None

- Returns

Boolean - True if there are no MES object links in the collection.

Code Examples

Code Snippet

```
#Prints False because there are three object links in the
collection.
mesObject = system.mes.loadMESObject('Vinegar', 'MaterialClass'
)
if mesObject != None:
    childList = mesObject.getChildCollection()
    print childList.isEmpty()
```

Output

```
False
```

size()

Description

Returns the number of MES object links in the collection. Depending if getParentCollection() or getChildCollection() is called to get the MESObjectCollection object, it will represent the number of parents or children.

Syntax

size()

- Parameters

None

- Returns

Integer - The number of MES object links in the collection.

Code Examples

Code Snippet

```
#In this example, Vinegar material class has got three
children. Therefore it prints 3.
mesObject = system.mes.loadMESObject('Vinegar', 'MaterialClass'
)
if mesObject != None:
    childList = mesObject.getChildCollection()
    print childList.size()
```

Output

```
3
```

Properties:

None

Code Examples

Code Snippet

```
#This example reads and prints the name of all child MES
objects that belong to the Vinegar Material Class.
mesObject = system.mes.loadMESObject('Vinegar', 'MaterialClass'
)

if mesObject != None:
    childList = mesObject.getChildCollection().getList()
    for child in childList:
```

```
        #The child variable is of the MESObjectLink type
        print child.getName()
```

Output

```
Balsamic Vinegar
Red Wine Vinegar
White Vinegar
```

getComplexProperty

Description

Get the collection of complex properties.

Method Options

getComplexProperty(complexPropertyName, entryName)

Syntax

getComplexProperty(complexPropertyName, entryName)

- Parameters

String complexPropertyName - The name for the type of complex property.

String entryName - The name of the complex property entry.

- Returns

The complex property collection.

- Scope

All

Code Examples

Code Snippet

```
seg = system.mes.createMESObject('ProcessSegment')
#Create a new material reference complex property
materialProp = seg.createComplexProperty('Material', 'Vinegar')
print seg.getComplexProperty('Material','Vinegar')
```

Output

```
{MaterialOptional=false, MaterialProductionSelectable=true,
MaterialUse=null, MaterialAutoGenerateLot=false, MaterialRef=,
MaterialRefUUID=null, MaterialRefType=null,
MaterialEquipmentRef=, MaterialEquipmentRefType=null,
MaterialEquipmentRefUUID=null, MaterialEnableSublots=false,
MaterialLotNoSource=null, MaterialLotNoSourceLink=null,
MaterialQuantitySource=null, MaterialQuantitySourceLink=null,
MaterialQuantity=null, MaterialUnits=null,
MaterialRatePeriod=null, MaterialRate=null,
MaterialCycleTime=0, MaterialFinalLotStatus=null,
MaterialAutoLotCompletion=Disabled,
MaterialLotDepletionWarning=0, MaterialLotStatusFilter=null}
```

getComplexProperty(complexPropertyName, index)

Syntax

getComplexProperty(complexPropertyName, index)

- Parameters

String complexPropertyName - The name for the type of complex property.

Integer index - The specified position in this list.

- Returns

The complex property collection.

- Scope

All

Code Examples

Code Snippet

```
seg = system.mes.createMESObject('ProcessSegment')
#Create a new material reference complex property
materialProp = seg.createComplexProperty('Material', 'Vinegar')
print seg.getComplexProperty('Material', 0)
```

Output

```
{MaterialOptional=false, MaterialProductionSelectable=true,
MaterialUse=null, MaterialAutoGenerateLot=false, MaterialRef=,
MaterialRefUUID=null, MaterialRefType=null,
MaterialEquipmentRef=, MaterialEquipmentRefType=null,
MaterialEquipmentRefUUID=null, MaterialEnableSublots=false,
MaterialLotNoSource=null, MaterialLotNoSourceLink=null,
MaterialQuantitySource=null, MaterialQuantitySourceLink=null,
MaterialQuantity=null, MaterialUnits=null,
MaterialRatePeriod=null, MaterialRate=null,
MaterialCycleTime=0, MaterialFinalLotStatus=null,
MaterialAutoLotCompletion=Disabled,
MaterialLotDepletionWarning=0, MaterialLotStatusFilter=null}
```

getComplexProperty(path)

Syntax

getComplexProperty(path)

- Parameters

MESPropertyPath The path to the MES property to get.

- Returns

The complex property for the specified path.

- Scope

All

Code Examples

Code Snippet

```
seg = system.mes.createMESObject('ProcessSegment')
#Create a new material reference complex property
materialProp = seg.createComplexProperty('Material', 'Vinegar')
print seg.getComplexProperty('Material.Vinegar')
```

Output

```
{MaterialOptional=false, MaterialProductionSelectable=true,
MaterialUse=null, MaterialAutoGenerateLot=false, MaterialRef=,
MaterialRefUUID=null, MaterialRefType=null,
MaterialEquipmentRef=, MaterialEquipmentRefType=null,
MaterialEquipmentRefUUID=null, MaterialEnableSublots=false,
MaterialLotNoSource=null, MaterialLotNoSourceLink=null,
MaterialQuantitySource=null, MaterialQuantitySourceLink=null,
MaterialQuantity=null, MaterialUnits=null,
MaterialRatePeriod=null, MaterialRate=null,
MaterialCycleTime=0, MaterialFinalLotStatus=null,
MaterialAutoLotCompletion=Disabled,
MaterialLotDepletionWarning=0, MaterialLotStatusFilter=null}
```

getComplexPropertyCount

Description

Get the number of complex properties.

Syntax

getComplexPropertyCount(complexPropertyName)

- Parameters

String complexPropertyName - The name for the complex property.

- Returns

Integer count - The size of the complex property collection.

- Scope

All

Code Examples

Code Snippet

```
#This code snippet will print the size of complex property
with name 'Material'
seg = system.mes.createMESObject('ProcessSegment')

#Create a new material reference complex property
materialProp = seg.createComplexProperty('Material', 'White
Vinegar')
materialProp = seg.createComplexProperty('Material', 'Balsamic
Vinegar')
materialProp = seg.createComplexProperty('Material', 'Red Wine
Vinegar')
print seg.getComplexPropertyCount('Material')
```

Output

```
3
```

getComplexPropertyItemNames

Description

Gets the list of production item with the specified complex property.

Syntax

getComplexPropertyItemNames(complexPropertyName)

- Parameters

String complexPropertyName - The name of the complex property.

- Returns

List<String> itemNameList - The list of production item names with the specified complex property.

- Scope

All

Code Examples

Code Snippet

```
segName = 'Mixed Nuts 8oz-Nuts Unlimited:Folsom:Packaging:
Packaging Line 1'
mesObject = system.mes.loadMESObject(segName, "OperationsSegmen
t")
prodList = mesObject.getComplexPropertyItemNames('ProductionSet
tings')
for item in prodList:
    print item, " - ", mesObject.getComplexProperty('Production
Settings',item).getModeRefProperty().getValue()
```

Output

```
Packaging Line 1 - Equipment Mode, Production
Packaging Line 1:Casepacker - Equipment Mode, Production
Packaging Line 1:Checkweigher - Equipment Mode, Production
Packaging Line 1:Filler - Equipment Mode, Production
Packaging Line 1:Palletizer - Equipment Mode, Production
Packaging Line 1:Labeler - Equipment Mode, Disabled
```

getComplexPropertyTypeNames

Description

Get the complex properties of a specific type.

Syntax

getComplexPropertyTypeNames()

- Parameters

None

- Returns

A list of complex properties that matches the given type.

- Scope

All

Code Examples

Code Snippet

```
#This example prints the list of complex properties that
matches the type 'Process Segment'.
seg = system.mes.loadMESObject('e9acd913-0ac2-497a-8c41-
efa9285bd3ed')

#Create a new material reference complex property
materialProp = seg.createComplexProperty('Material', 'Vinegar')
print seg.getComplexPropertyTypeNames()
```

Output

```
[Material, Equipment, SupplEquip, Personnel]
```

getCustomProperties

Description

Returns the custom properties for the definition object.

Syntax

getCustomProperties ()

- Parameters

None

- Returns

MESPropertyCollection - The list of custom properties.

- Scope

All

Code Examples

Code Snippet

```
matDef = system.mes.loadMESObject('Cane Sugar', 'MaterialDef')
print matDef.getCustomProperties()
```

getCustomPropertiesFull

Description

Return all the custom properties definitions of an MES object. The results are returned as name (or property path) definition pairs. This is useful for copying custom properties from one MES object to another MES object that doesn't have the custom properties defined.

Method Options

getCustomPropertiesFull()

Description

Return the custom properties for the MES object as a Python dictionary. The Python dictionary will contain custom property path, definition pairs. The definition is a Python list containing the data type, value, description, units, production visible and required settings of the custom property.

Because custom properties can be nested, the path of the custom property is used. Each level of the path is separated with a period. This allows for names to be duplicated provided the path is unique. For example, it is possible for a MES object to have custom properties "Dimension 1.Wdith" and "Dimension 2.Width"

Syntax

getCustomProperties()

- Parameters

Nothing

- Returns

PyDictionary - A Python dictionary containing custom property path (name) definition pairs.

- Scope

All

Code Examples

Code Snippet

```
#Get the full custom property definitions for a MES object
cp = obj.getCustomPropertiesFull()
#Print the Python dictionary of all custom properties
print cp
#Get the definition for just the Width custom property
widthDef = cp['Dimension.Width']
#Print the Python list for the Width custom property definition
```

```
print widthDef
#Print setting individually for the Width custom property
print widthDef[0]    #Data type
print widthDef[1]    #Value
print widthDef[2]    #Description
print widthDef[3]    #Units
print widthDef[4]    #Production Visible
print widthDef[5]    #Required
```

Output

```
{u'Dimension.Width': [u'Int8', u'10', u'Box width', u'in',
True, False], u'Dimension': [u'String', u'', u'', u'', True,
False], u'Dimension.Height': [u'Int8', u'12', u'Box height',
u'in', True, False]}
[u'Int8', u'10', u'Box width', u'in', True, False]
Int8
10
Box width
in
True
False
```

getCustomPropertiesFull(complexPropertyName, entryName)

Description

Return the custom properties for a complex property of a MES object as a Python dictionary. When the complex properties on a MES object (for example, a material reference) have custom properties, this method is used to get them by type and name. The Python dictionary will contain custom property path, definition pairs. The definition is a Python list containing the data type, value, description, units, production visible and required settings of the custom property.

Because custom properties can be nested, the path of the custom property is used. Each level of the path is separated with a period. This allows for names to be duplicated provided the path is unique. For example, it is possible for a MES object to have custom properties "Dimension 1.Wdith" and "Dimension 2.Width"

Syntax

getCustomPropertiesFull(complexPropertyName, entryName)

- Parameters

String complexPropertyType - The name for the type of complex property.

String name - The name of the complex property entry.

- Returns

PyDictionary - A Python dictionary containing custom property path (name) definition pairs.

- Scope

All

Code Examples

Code Snippet

```
#Load a segment MES object
proSeg = system.mes.loadMESObject('Receive Steel', 'ProcessSegm
ent')
#Get the custom property values for the Steel In material
reference
cp = proSeg.getCustomPropertiesFull('Material', 'Steel In')
#Get the definition for just the Thickness custom property
cpDef = cp['Dimension.Thickness']
#Print the Python list for the Thickness custom property
definition
print cpDef
#Print setting individually for the Thickness custom property
print cpDef [0]    #Data type
print cpDef [1]    #Value
print cpDef [2]    #Description
print cpDef [3]    #Units
print cpDef [4]    #Production Visible
print cpDef [5]    #Required
```

Output

```
[u'Float8', u'100', u'Thickness of steel', u'1/1000th', True,
True]
Float8
100
```

```
Thickness of steel
1/1000th
True
True
```

getCustomPropertyDescription

Description

Get the description of the custom property by name or property path.

Syntax

getCustomPropertyDescription(propertyPath)

- Parameters

String propertyPath - The name or property path of the custom property to get the description for.

- Returns

String description - The custom property description.

- Scope

All

Code Examples

Code Snippet

```
#Load an MES object
obj = system.mes.loadMESObject('Box', 'MaterialDef')

#Add a custom property
obj.addCustomProperty('Kind', 'String', 'Kind of box', '', True
, True)
```

```
#Print the current enabled state
print obj.getCustomPropertyDescription('Kind')
```

Output

```
Kind of box
```

getCustomPropertyEnabled

Description

Get the enabled state of a custom property. If the enabled state is True then custom property is available. If False, the custom property will not appear in component, be used during production or is not accessible in script.

Syntax

getCustomPropertyEnabled(propertyPath)

- Parameters

String propertyPath - The name or property path of the custom property to get the enabled state.

- Returns

Boolean The enabled state of the custom property.

- Scope

All

Code Examples

Code Snippet

```
#Load an MES object
obj = system.mes.loadMESObject('Box', 'MaterialDef')

#Add a custom property
obj.addCustomProperty('Kind', 'String')

#Print the current enabled state
print obj.getCustomPropertyEnabled('Kind')

#Disable the custom property named Kind
obj.setCustomPropertyEnabled('Kind', False)

#Print the current enabled state
print obj.getCustomPropertyEnabled('Kind')
```

```
Output

True
False
```

getCustomPropertyUnits

Description

Get the units of the custom property.

Syntax

getCustomPropertyUnits(propertyPath)

- Parameters

String propertyPath - The name or property path of the custom property to get the description for.

- Returns

String units - The units of the new custom property.

- Scope

All

Code Examples

Code Snippet

```
#Load MES object
obj = system.mes.loadMESObject('Box', 'MaterialDef')
print obj.getCustomPropertyUnits('Kind.Class of Box')
```

Output

```
lbs
```

getCustomPropertyValues

Description

Return all the custom properties off an MES object as name (or property path) value pairs. This simplifies copying custom property values from one MES object to another.

Method Options

getCustomPropertyValues()

Description

Return the custom properties for the MES object as a Python dictionary. The Python dictionary will contain custom property path, value pairs. Because custom properties can be nested, the path of the custom property is used. Each level of the path is separated with a period. This allows for names to be duplicated provided the path is unique. For example, it is possible for a MES object to have custom properties "Dimension 1.Wdith" and "Dimension 2.Width"

Syntax

getCustomProperties()

- Parameters

Nothing

- Returns

PyDictionary - A Python dictionary containing custom property path (name) value pairs.

- Scope

All

Code Examples

Code Snippet

```
#Get the Power Supply material class object
matCls = system.mes.loadMESObject('Power Supply', 'MaterialClas
s')
#Get the custom property values for the object
cp = matCls.getCustomPropertyValues()
#Cycle through and print the custom property values
for path in cp.keys():
    print 'Path: %s = %s' % (path, cp[path])
print
#Just access a specific custom property by known path
print 'Dimension 1.Width = %s' % cp['Dimension 1.Width']
```

Output

```
Path: Dimension 1.Height = 340
Path: Dimension 2 = None
Path: Dimension 2.Height = 341
Path: Dimension 1 = None
Path: Dimension 2.Width = 891
Path: Dimension 1.Width = 890
Dimension 1.Width = 890
```

getCustomPropertyValues(complexPropertyType, String name)

Description

Return the custom properties for the complex property (resource reference) as a Python dictionary. The Python dictionary will contain custom property path, value pairs. Because custom properties can be nested, the path of the custom property is used. Each level of the path is separated with a period. This allows for names to be duplicated provided the path is unique. For example, it is possible for a MES object to have custom properties "Dimension 1.Wdith" and "Dimension 2.Width".

For response segments, references to resources are created for each lot, person or supplemental equipment. For example, if a segment is started filling tank 1 and then switches to tank 2, then there will be two complex properties. One that has all the information while tank 1 was selected and a second that has all the information while tank 2 was selected. Custom properties can be added to the complex properties that will be unique for the time each tank was selected. If a pH reading is collected for each tank, then the separate readings can be saved in the appropriate complex property. To specify which complex property to access, extended naming is used. Referring to our sample, if we named the material complex property "Liquid", then the extend name for the first entry is "Liquid-1" and the second entry is "Liquid-2".

Syntax

getCustomPropertyValues(complexPropertyType, String name)

- Parameters

String complexPropertyType - The name for the type of complex property.

String name - The name of the complex property entry.

- Returns

PyDictionary - A Python dictionary containing custom property path (name) value pairs.

- Scope

All

Code Examples

```
Code Snippet
```

```
#Get the Storage Tanks equipment class object
proSeg = system.mes.loadMESObject('Receive Steel', 'ProcessSegm
ent')
#Get the custom property values for the Steel In material
reference
cp = proSeg.getCustomPropertyValues('Material', 'Steel In')
#Cycle through and print the custom property values
for path in cp.keys():
    print 'Path: %s = %s' % (path, cp[path])
print
#Just access a specific custom property by known path
print 'Dimension 1.Width = %s' % cp['Dimension 1.Width']
```

getMESObjectType

Description

Get the type of the MES object. This returns an object that represents the MES object type. This object contains other information associated with the type. In most cases only the name of the MES object type is needed and using the getMESObjectTypeName script function is recommended instead.

Syntax

getMESObjectType()

- Parameters

None

- Returns

MESObjectType - The type of MES object.

- Scope

All

Code Examples

Code Snippet

```
#This example will print the type of MES object.
seg = system.mes.loadMESObject('e9acd913-0ac2-497a-8c41-
efa9285bd3ed')

#Create a new material reference complex property
materialProp = seg.createComplexProperty('Material', 'Vinegar')
print seg.getMESObjectType()
```

Output

```
 Process Segment
```

getMESObjectTypeName.

Description

Get the name of the MES object type.

Syntax

getMESObjectTypeName()

- Parameters

None

- Returns

String - The MES object type name of the MES object.

- Scope

All

Code Examples

Code Snippet

```
obj = system.mes.loadMESObject('Box', 'MaterialDef')
print obj.getMESObjectTypeName()
```

Output

```
Material Definition
```

getName.

Description

Get the name of the MES object. For each type of MES object, the name must be unique. For example, the name of each material definition is unique. In addition, the MES object name must also be unique for various categories of MES object types.

Syntax

getName()

- Parameters

None

- Returns

String - MES object name.

- Scope

All

Info

Below are the categories of MES object types that the names will be unique:

Category	MES Object Types
Equipment	Equipment, EquipmentClass
Material, Personnel	MaterialDef, MaterialClass, Person, Personnel Class
Operation	Operations Definition, OperationSegment

Code Examples

Code Snippet

```
obj = system.mes.loadMESObject('Box', 'MaterialDef')
print obj.getName()
```

Output

```
Box
```

getParentCollection

Description

Get the collection of parents for an MES object.

Syntax

getParentCollection()

- Parameters

None

- Returns

Returns a collection object containing references to all parents of the MES object.

- Scope

All

MESObjectCollection Details

A MESObjectCollection object is used by MES objects to hold parents and children. Normally, the parent and child script functions of the MES objects should be used, but this is provided as a reference to the MESObjectCollection object itself and provides some additional functionality.

Methods:

get(uuid)

Description

Returns the MES object link for the specified UUID. If the specified UUID does not exist, None will be returned.

Syntax

get(uuid)

- Parameters

None

- Returns

String mesObjectLink - The MES object link corresponding to the uuid.

Code Examples

Code Snippet

```
#Object link corresponding to the specified uuid is returned.
```

```
mesObject = system.mes.loadMESObject('Vinegar', 'MaterialClass'
)
if mesObject != None:
    childList = mesObject.getChildCollection()
    print childList.get('5acf3c9f-2789-44af-888f-fce08d9972a7')
```

Output

```
Red Wine Vinegar
```

getList()

Description

Returns a list of MES object links. Depending if getParentCollection() or getChildCollection() is called to get the MESObjectCollection object, it will contain MES object links that are parents or children.

Syntax

getList()

- Parameters

None

- Returns

String mesObjectLinkList - A list containing MES object links.

Code Examples

Code Snippet

```
#This example reads the child MES object links that belong to
the Vinegar Material Class.
```

```
mesObject = system.mes.loadMESObject('Vinegar', 'MaterialClass'
)
if mesObject != None:
    childList = mesObject.getChildCollection().getList()
    for child in childList:
        print child.getName()
```

Output

```
Balsamic Vinegar
Red Wine Vinegar
White Vinegar
```

isEmpty()

Description

Returns True if no MES object links exist in the collection.

Syntax

isEmpty()

- Parameters

None

- Returns

Boolean - True if there are no MES object links in the collection.

Code Examples

Code Snippet

```
#Prints False because there are three object links in the
collection.
```

```
mesObject = system.mes.loadMESObject('Vinegar', 'MaterialClass'
)
if mesObject != None:
    childList = mesObject.getChildCollection()
    print childList.isEmpty()
```

Output

```
False
```

size()

Description

Returns the number of MES object links in the collection. Depending if getParentCollection() or getChildCollection() is called to get the MESObjectCollection object, it will represent the number of parents or children.

Syntax

size()

- Parameters

None

- Returns

Integer - The number of MES object links in the collection.

Code Examples

Code Snippet

```
#In this example, Vinegar material class has got three
children. Therefore it prints 3.
```

```
mesObject = system.mes.loadMESObject('Vinegar', 'MaterialClass'
)
if mesObject != None:
    childList = mesObject.getChildCollection()
    print childList.size()
```

Output

```
3
```

Properties:

None

Code Examples

Code Snippet

```
#This example reads and prints the name of all parent MES
objects that belong to Balsamic Vinegar.
mesObject = system.mes.loadMESObject('Balsamic Vinegar', 'Mater
ialDef')

if mesObject != None:
    parentList = mesObject.getParentCollection().getList()
    for parent in parentList:

        #The parent variable is of the MESObjectLink type
        print parent.getName()
```

Output

```
Vinegar
```

getPropertyValue

Description

Get a property value of an MES object by name or path. The property can be a core property, custom property or complex property of the MES object. In the case where custom properties are nested, a path is required to return the correct value.

The type of the value depends on the property being read. If no value is currently assigned to the property, None will be returned.

Syntax

getPropertyValue(propertyPath)

- Parameters

String propertyPath - The name or path of the property being read.

- Returns

The value of the property. The type depends on the property being read.

- Scope

All

Code Examples

Code Snippet

```
#Load a MES object
obj = system.mes.loadMESObject('Box', 'MaterialDef')

#Read and print the Width and Height custom properties that
are children of the Dimension custom property.
width = obj.getPropertyValue('Dimension.Width')
height = obj.getPropertyValue('Dimension.Height')
print "Width = %d, Height = %d" % (width, height)
```

Output

```
Width = 10, Height = 12
```

getUUID

Description

Return the UUID value from the UUID property of the MES object. UUID stands for Universally Unique Identifier and each MES object will have a UUID assigned.

Syntax

getUUID()

- Parameters

None

- Returns

String - UUID of the MES object

- Scope

All

Code Examples

Code Snippet

```
obj = system.mes.loadMESObject('Box', 'MaterialDef')
print obj.getUUID()
```

Output

```
b17159b6-81ce-4057-a5b3-626126079320
```

getVersion

Description

Return the version number of the MES object. Not all MES objects have version number. Every time a definition type of MES object is modified, the version number is increased. When new schedules or production is run based on the definition MES object, it is tied the latest version of the definition MES objects.

Syntax

getVersion()

- Parameters

None

- Returns

Integer - The version number of this MES object.

- Scope

All

Code Examples

Code Snippet

```
obj = system.mes.loadMESObject('Box', 'MaterialDef')
print obj.getVersion()
```

Output

```
3
```

isEnabled

Description

Get the enabled state of the MES object. If an MES object is not enabled, it will not show in any selection list or able to be selected within script. Essentially, the enabled state will change to false when the MES object is deleted. Deleted MES object still reside in the system for history of past production.

Syntax

isEnabled()

- Parameters

None

- Returns

Boolean - The enabled state of the MES object.

- Scope

All

Code Examples

Code Snippet

```
obj = system.mes.loadMESObject('Box', 'MaterialDef')
print obj.isEnabled()
```

Output

```
True
```

isModified

Description

If one or more properties of an MES object are changed, then True will be returned.

Syntax

isModified()

- Parameters

None

- Returns

Boolean - The MES object modified state.

- Scope

All

Code Examples

Code Snippet

```
#Load a MES object
obj = system.mes.loadMESObject('Box', 'MaterialDef')
print obj.isModified()

#Change a property of it
obj.setPropertyValue('Name', 'Empty Box')
print obj.isModified()
```

Output

```
False
True
```

removeChild

Description

Remove another MES child object for an MES object.

MES objects can have children and parents. Depending on the type of MES object determines the types of children or parents that can be added. For example, a material class (MESMaterialClass) object can have material definitions (MESMaterialDef) as children, but material definitions objects cannot have material class objects as children.

Note

When a child is removed from an MES object, then the parent reference will be removed from the child behind the scenes. Likewise, when a parent is removed from a child, the child reference will be removed from the parent behind the scenes. This insures if integrity of relationships are maintained.

Syntax

removeChild(mesObject)

- Parameters

AbstractMESObject mesObject - An instance to an MES object. An AbstractMESObject object is just the generic form of an MES object when the specific type is unknown.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Get the Screws material class object
matCls = system.mes.loadMESObject('Screws', 'MaterialClass')

#Get the 10-32 NC Screw material definition object
matDef = system.mes.loadMESObject('10-32 NC Screw', 'MaterialDe
f')

#Remove the 10-32 NC Screw material definition object from the
Screws material class object
matCls.removeChild(matDef)

#Save the changes
system.mes.saveMESObject(matCls)
```

removeComplexProperty

Description

Removes an existing complex property.

Syntax

removeComplexProperty(complexPropertyName, entryName)

- Parameters

String complexPropertyName - The name for the type of complex property.

String entryName - The name of the complex property entry.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
seg = system.mes.createMESObject('ProcessSegment')

#In order to make it more clear, Let us create a complex
property named Equipment.
materialProp = seg.createComplexProperty('Equipment', 'Vinegar
Tank1')

#This code snippet removed the complex property named
'Equipment'. So that when script function getComplexProperty()
is executed it prints None.
removeProp = seg.removeComplexProperty('Equipment', 'Vinegar
Tank1')
print seg.getComplexProperty('Equipment', 'Vinegar Tank1')
```

Output

```
None
```

removeParent

Description

Remove another MES parent object for an MES object.

MES objects can have children and parents. Depending on the type of MES object determines the types of children or parents that can be added. For example, a material class (MESMaterialClass) object can have material definitions (MESMaterialDef) as children, but material definitions objects cannot have material class objects as children.

Note

When a child is removed from an MES object, then the parent reference will be removed from the child behind the scenes. Likewise, when a parent is removed from a child, the child reference will be removed from the parent behind the scenes. This insures the integrity of relationships are maintained.

Syntax

removeParent(mesObject)

- Parameters

AbstractMESObject mesObject - An instance to an MES object. An AbstractMESObject object is just the generic form of an MES object when the specific type is unknown.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Get the 10-32 NC Screw material definition object
matDef = system.mes.loadMESObject('10-32 NC Screw', 'MaterialDe
f')

#Get the Screws material class object
matCls = system.mes.loadMESObject('Screws', 'MaterialClass')

#Remove the Screws material class object from the 10-32 NC
Screw material definition object
matDef.removeParent(matCls)

#Save the changes
system.mes.saveMESObject(matDef)
```

renameComplexProperty

Description

Renaming an existing complex property.

Syntax

renameComplexProperty(complexPropertyName, entryName, newEntryName)

- Parameters

String complexPropertyName - The name for the type of complex property.

String entryName - The name of the complex property entry.

String newEntryName - The new complex property name.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
seg = system.mes.createMESObject('ProcessSegment')

#Creates a complex property with name 'Equipment'
proSeg = seg.createComplexProperty('Equipment', 'Vinegar Tank1'
)

#Changes the name 'Vinegar Tank1' into 'Vinegar Tank2'
renameProp = seg.renameComplexProperty('Equipment', 'Vinegar
Tank1', 'Vinegar Tank2')
```

renameCustomProperty

Description

Rename an existing custom property.

Syntax

renameCustomProperty(propertyPath, newName)

- Parameters

String propertyPath - The name or property path of the existing custom property to rename.

String newName **-** New custom property name.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Load an MES object
obj = system.mes.loadMESObject('Box', 'MaterialDef')

#Add a custom property named Type
obj.addCustomProperty('Type', 'String')

#Remember to save the MES object
#system.mes.saveMESObject(obj)

#Print it to see the name of the new custom property
print obj.getCustomPropertiesFull()

#Rename the custom property from Type to Kind
obj.renameCustomProperty('Type', 'Kind')
```

```
#Print it to see the new custom property name
print obj.getCustomPropertiesFull()

#Remember to save the MES object
system.mes.saveMESObject(obj)
```

Output

```
{u'Type': [u'String', u'', u'', u'', False, False]}
{u'Kind': [u'String', u'', u'', u'', False, False]}
```

setCustomPropertyDescription

Description

Set the description of the custom property by name or property path.

Syntax

setCustomPropertyDescription(propertyPath, description)

- Parameters

String propertyPath - The name or property path of the custom property to get the description for.

String description - The custom property description.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Load an MES object
obj = system.mes.loadMESObject('Box', 'MaterialDef')

#Set a new description for Kind
obj.setCustomPropertyDescription('Kind', 'Class of Box')

#Prints the current description of the custom property.
print obj.getCustomPropertyDescription('Kind')
```

Output

```
Class of Box
```

setCustomPropertyEnabled

Description

Set the enabled state of a custom property. If set to True then custom property is available. If False, the custom property will not appear in component, be used during production or is not accessible in script.

Syntax

setCustomPropertyEnabled(propertyPath, enable)

- Parameters

String propertyPath - The name or property path of the custom property to set the enabled state.

Boolean enable - The enabled state to set the custom property to.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Load an MES object
obj = system.mes.loadMESObject('Box', 'MaterialDef')

#Add a custom property
obj.addCustomProperty('Kind', 'String')

#Print the current enabled state
print obj.getCustomPropertyEnabled('Kind')

#Disable the custom property named Kind
obj.setCustomPropertyEnabled('Kind', False)

#Print the current enabled state
print obj.getCustomPropertyEnabled('Kind')
```

Output

```
True
False
```

setCustomPropertyUnits

Description

Set the units of the custom property.

Syntax

setCustomPropertyUnits(propertyPath, units)

- Parameters

String propertyPath - The name or property path of the custom property to get the description for.

String units - The units of the new custom property.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Load MES object
obj = system.mes.loadMESObject('Box', 'MaterialDef')
obj.setCustomPropertyUnits('Kind.Class of Box','lbs')
print obj.getCustomPropertyUnits('Kind.Class of Box')
```

Output

```
lbs
```

setCustomPropertyValues

Description

This method can be used to set custom properties values or create custom properties of an MES object. The format of the customProperties parameter determines if just custom property values will be assigned or custom properties will be created and values assigned.

Because custom properties can be nested, the path of the custom property is used. Each level of the path is separated with a period. This allows for names to be duplicated provided the path is unique. For example, it is possible for a MES object to have custom properties "Dimension 1.Width" and "Dimension 2.Width"

Python dictionary

To just set the custom property values, use a Python dictionary for the customProperties parameter that contains name value pairs.

To create custom properties, use a Python dictionary for the customProperties parameter that contains name definition pairs. The custom property definition is a Python list that contains the settings. The order of the setting must be **data type**, **value** , **description**, **units**, **production visible** and **required**. Optionally, the production visible and required can be left out. If they are left out the default values of production visible = true and required = false will be used.

Syntax

setCustomPropertyValues(customProperties)

- Parameters

PyDictionary customProperties - A Python dictionary containing custom property path (name) definition pairs.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Load a material lot MES object
obj = system.mes.loadMESObject('0000000031', 'MaterialLot')

#Build the definition of the custom properties to add
current_dataType = 'Float8'
current = 3.2
current_desc = 'Test results - current'
current_units = 'Amps'
```

```
volts_dataType = 'Float8'
volts= 12.0
volts_desc = 'Test results - current'
volts_units = 'Volts'
lotCP = {'Amps' : [current_dataType , current, current_desc ,
current_units], 'Volts': [volts_dataType , volts, volts_desc,
volts_units]}
print lotCP

#Add the custom properties to the material lot MES object
obj.setCustomPropertyValues(lotCP)

#Remember to save the MES object
system.mes.saveMESObject(obj)
```

Output

```
{'Amps': ['Float8', 3.2, 'Test results - current', 'Amps'],
'Volts': ['Float8', 12.0, 'Test results - volts', 'Volts']}
```

Code Snippet

```
#Copy custom properties from one MES object to another
#Load a material lot MES object to copy custom properties
values from
obj1 = system.mes.loadMESObject('0000000031', 'MaterialLot')

#Read the custom properties
cp = obj1.getCustomPropertiesFull()
print cp

#Load a material lot MES object to copy custom properties to
obj2 = system.mes.loadMESObject('0000000030', 'MaterialLot')

#Add the custom properties
obj2.setCustomPropertyValues(cp)

#Change the values of the custom properties
current = 3.4
volts = 11.9
obj2.setPropertyValue('Amps', str(current))
obj2.setPropertyValue('Volts', str(volts))

#Remember to save the MES object
system.mes.saveMESObject(obj2)
```

Output

```
{u'Amps': [u'Float8', u'3.2', u'Test results - current',
u'Amps', True, False], u'Volts': [u'Float8', u'12.0', u'Test
results - volts', u'Volts', True, False]}{u'Amps': 3.4,
u'Volts': 11.9}
```

Code Snippet

```
#Load a response segment MES object
segObj = system.mes.loadMESObject('8163976b-1333-4aa2-9f76-
e622ef1b5174')

#Build the custom property definitions
weight = 25
length = 152
cp = {'Weight': ['Int4', weight], 'Length': ['Int4', length]}

#Add the custom properties to the Pencil material reference
segObj.setCustomPropertyValues('ResponseMaterial', 'Pencil',
cp)  .

print segObj.getCustomPropertyValues('ResponseMaterial', 'Penci
l')

#Remember to save the MES object
system.mes.saveMESObject(segObj)
```

Output

```
{u'ResponseMaterial.Pencil.Weight': 25, u'ResponseMaterial.
Pencil.Length': 152}
```

setEnabled

Description

Set the enabled state of the MES object. By using this script function with parameter of False will delete the MES object. After setting the enabled state to false, the object must be save for the changes to take effect. At the time the disabled MES object is saved, the name is also modified. This allows for the name to be reused in the future without naming conflicts.

Syntax

setEnabled(enable)

- Parameters

Boolean enable - The new enabled state to make the MES object.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Disable MES object example
#Load the MES object
obj = system.mes.loadMESObject('Box', 'MaterialDef')
print "Name of enabled MES object: %s" % obj.getName()

#Disable the MES object and save it
obj.setEnabled(False)
system.mes.saveMESObject(obj)
```

Output

```
Name of enabled MES object: Box
```

Code Snippet

```
#Enable MES object example
#Load the disabled MES object
obj = system.mes.loadDisabledMESObject('Box', 'MaterialDef')
print "Name of disabled MES object: %s" % obj.getName()

#Disable the MES object and save it
obj.setEnabled(True)
system.mes.saveMESObject(obj)
```

Output

```
Name of disabled MES object: Box{9320}
```

setPropertyValue

Description

Set a property value of an MES object by name or path. The property can be a core property, custom property or complex property of the MES object. In the case where custom properties are nested, a path is required to set the correct value.

The property value is passed as a String, but is converted to the correct data type for the property.

Syntax

setPropertyValue(propertyPath, value)

- Parameters

String propertyPath - The name or path of the property being written to.

String value - The value to set the property to.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Load a MES object
obj = system.mes.loadMESObject('Box', 'MaterialDef')

#Get and print the values before changing them
width = obj.getPropertyValue('Dimension.Width')
height = obj.getPropertyValue('Dimension.Height')
print "Values before setPropertyValue is called: Width = %d,
Height = %d" % (width, height)

#Set the properties to new values
width = 14
height = 24
obj.setPropertyValue('Dimension.Width', str(width))
obj.setPropertyValue('Dimension.Height', str(height))

#Don't forget to save the MES object after changing property
values
system.mes.saveMESObject(obj)

#Get and print the new values of the properties
width = obj.getPropertyValue('Dimension.Width')
height = obj.getPropertyValue('Dimension.Height')
print "Values after setPropertyValue is called: Width = %d,
Height = %d" % (width, height)
```

Output

```
Values before setPropertyValue is called: Width = 10, Height =
12
Values after setPropertyValue is called: Width = 14, Height =
24
```

Code Snippet

```
#Rename MES object example
```

```
#Load a MES object
obj = system.mes.loadMESObject('Box', 'MaterialDef')

#Get and print the current name of the MES object
#Notice either method can be used to read the Name property
print obj.getName()
print obj.getPropertyValue('Name')

#Change the name of the MES object
obj.setPropertyValue('Name', 'Empty Box')

#Don't forget to save the MES object after changing property
values
system.mes.saveMESObject(obj)
```

Methods

getMESObjectType()

Description

Gets the MES object type that this property is set to.

Syntax

getMESObjectType()

- Parameters

None

- Returns

MESObjectTypes type - The MES object type that this property is set to.

- Scope

All

getParentMESObjectUUID()

Description

Gets the uuid of the parent object for this property.

Syntax

getParentMESObjectUUID()

- Parameters

None

- Returns

String parentMESObjectUUID - The uuid of the parent object.

- Scope

All

getReferenceMESObjectName()

Description

Gets the name of the reference object of this property.

Syntax

getReferenceMESObjectName()

- Parameters

None

- Returns

String name - The name of the reference object of this property.

- Scope

All

getReferenceObject(mesObjectTypeName, mesObjectUUID)

Description

Gets the reference object for this property.

Syntax

getReferenceObject(mesObjectTypeName, mesObjectUUID)

- Parameters

String mesObjectTypeName - The name of the type of MES object to return reference object for.

String mesObjectUUID - The UUID of MES object to return the reference object for. See UUIDs for more information.

- Returns

AbstractMESObject - The reference object for this property.

- Scope

All

getReferenceOptions(referenceType, searchPattern)

Description

Gets the reference options for this property.

Syntax

getReferenceOptions(referenceType, searchPattern)

- Parameters

MESObjectTypes referenceType - The reference of MES object type that this property is set to.

String searchPattern - The search pattern to filter the results by. It can contain the * and ? wild card characters.

- Returns

List <MESObjectLink> - The reference options for this property.

- Scope

All

getRefPropertyPath()

Description

Gets the reference property path for this property.

Syntax

getRefPropertyPath()

- Parameters

None

- Returns

String refPropertyPath - The reference property path for this property.

- Scope

All

getVersionRefUUID()

Description

Gets the version reference uuid for this property.

Syntax

getVersionRefUUID()

- Parameters

None

- Returns

String versionRefUUID - The version reference uuid for this property.

- Scope

All

hasParentMESObjectUUID()

Description

Checks whether there is a parent object uuid associated with this property.

Syntax

hasParentMESObjectUUID()

- Parameters

None

- Returns

Boolean - True if there exist a parent object uuid and False otherwise.

- Scope

All

hasRefPropertyPath()

Description

Checks whether there is a reference property path for this property.

Syntax

hasRefPropertyPath()

- Parameters

None

- Returns

Boolean - True if there exist a reference property path and False otherwise.

- Scope

All

setParentMESObjectUUID(parentMESObjectUUID)

Description

Sets the uuid of the parent object for this property.

Syntax

setParentMESObjectUUID(parentMESObjectUUID)

- Parameters

String parentMESObjectUUID - The uuid of the parent object.

- Returns

Nothing

- Scope

All

setRefPropertyPath(refPropertyPath)

Description

Sets the reference property path for this property.

Syntax

setRefPropertyPath(refPropertyPath)

- Parameters

String refPropertyPath - The reference property path for this property.

- Returns

Nothing

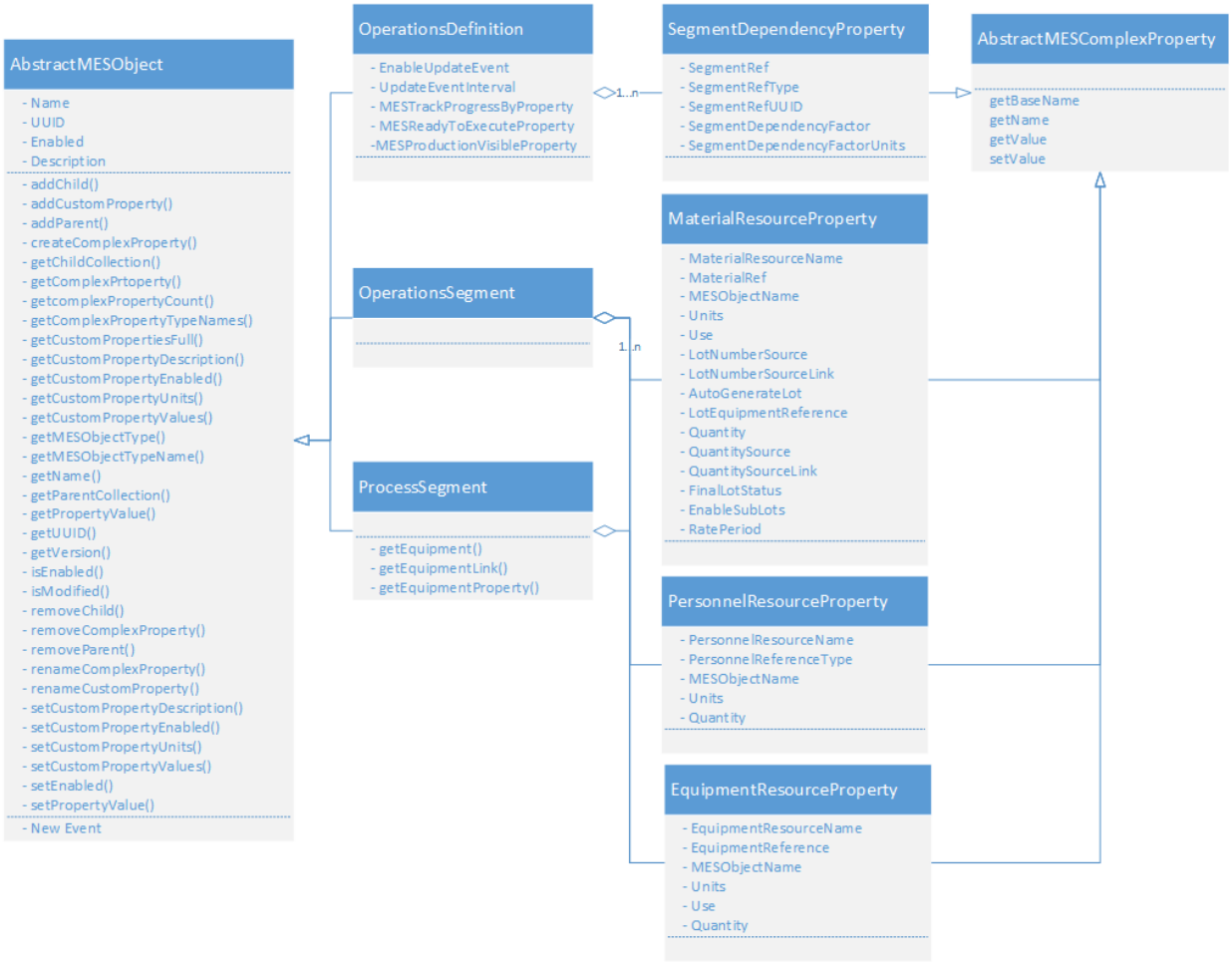
- Scope

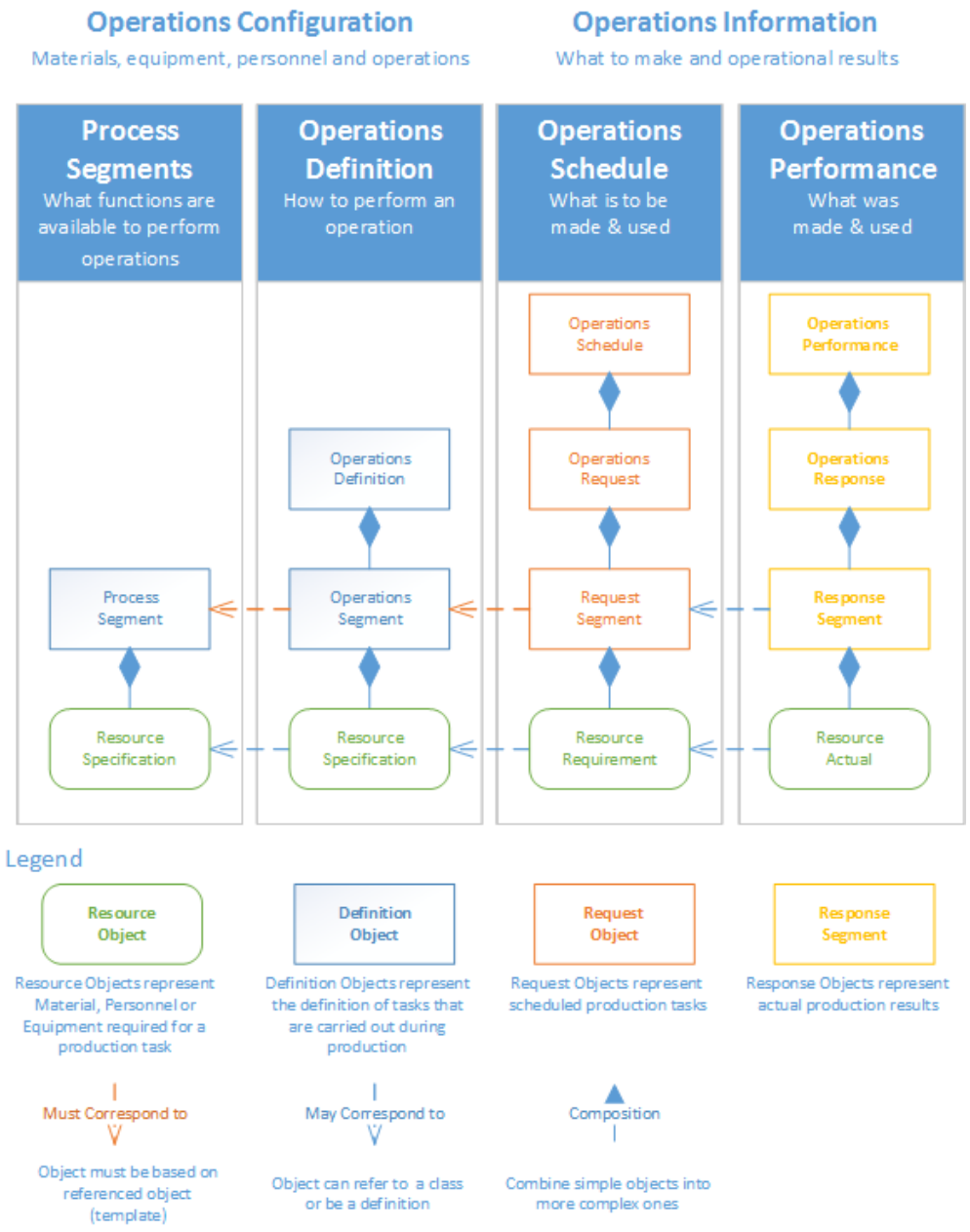
All

Definition Objects

Definition Objects as defined by ISA-95 represent the definition of task that are carried out during production. There are three object types that fall under this category, **Process Segment** , **Operation Segment** and **Operation Definition**.

Each of these objects inherits the AbstractMESObject properties and methods.


AbstractMESObject
- Name
- UUID
- Enabled
- Description
- addChild()
- addCustomProperty()
- addParent()
- createComplexProperty()
- getChildCollection()
- getComplexPrtoperty()
- getcomplexPropertyCount()
- getComplexPropertyTypeNames()
- getCustomPropertiesFull()
- getCustomPropertyDescription()
- getCustomPropertyEnabled()
- getCustomPropertyUnits()
- getCustomPropertyValues()
- getMESObjectType()
- getMESObjectTypeName()
- getName()
- getParentCollection()
- getPropertyValue()
- getUUID()
- getVersion()
- isEnabled()
- isModified()
- removeChild()
- removeComplexProperty()
- removeParent()
- renameComplexProperty()
- renameCustomProperty()
- setCustomPropertyDescription()
- setCustomPropertyEnabled()
- setCustomPropertyUnits()
- setCustomPropertyValues()
- setEnabled()
- setPropertyValue()
- New Event
OperationsDefinition
- EnableUpdateEvent
- UpdateEventInterval
- MESTrackProgressByProperty
- MESReadyToExecuteProperty
-MESProductionVisibleProperty
1..n
SegmentDependencyProperty
- SegmentRef
- SegmentRefType
- SegmentRefUUID
- SegmentDependencyFactor
- SegmentDependencyFactorUnits
AbstractMESComplexProperty
getBaseName
getName
getValue
setValue
OperationsSegment
1..n
MaterialResourceProperty
- MaterialResourceName
- MaterialRef
- MESObjectName
- Units
- Use
- LotNumberSource
- LotNumberSourceLink
- AutoGenerateLot
- LotEquipmentReference
- Quantity
- QuantitySource
- QuantitySourceLink
- FinalLotStatus
- EnableSubLots
- RatePeriod
ProcessSegment
- getEquipment()
- getEquipmentLink()
- getEquipmentProperty()
PersonnelResourceProperty
- PersonnelResourceName
- PersonnelReferenceType
- MESObjectName
- Units
- Quantity
EquipmentResourceProperty
- EquipmentResourceName
- EquipmentReference
- MESObjectName
- Units
- Use
- Quantity



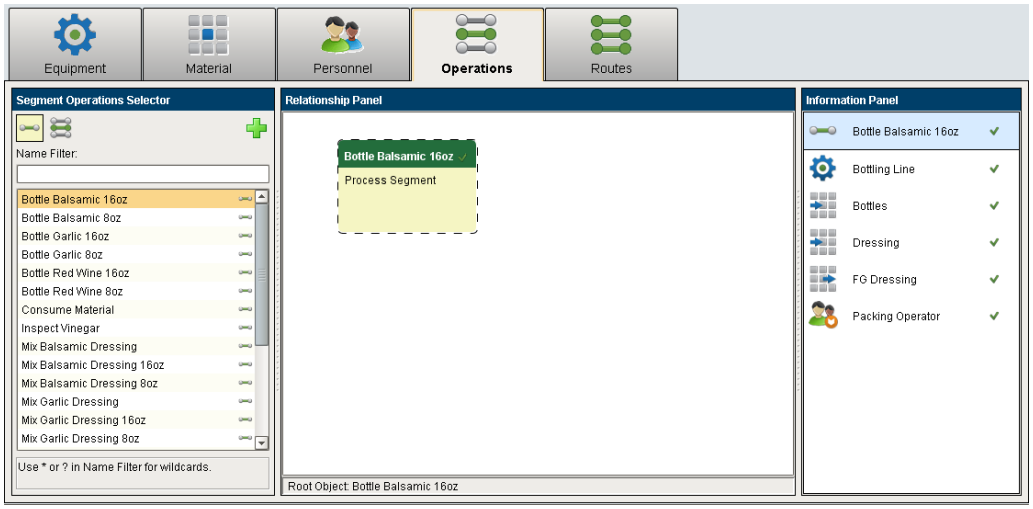
Process Segment

The Process Segment is an object used to hold the definition of basic tasks that are performed in a manufacturing facility. Tasks can be as simple as 'Produce Product' or 'Unload Material', but can be more granular like 'Changeover Line', 'Clean', 'Lab Inspection', 'Heat Up', 'Pre-Production', 'Produce Product' and 'Run Out Line'.

The Process Segment is never used in an actual operation, but acts as a template for Operation Segments to be created and based against.

This object inherits the AbstractMESObject properties and methods and extends it with some custom methods.

Goto Process Segment section

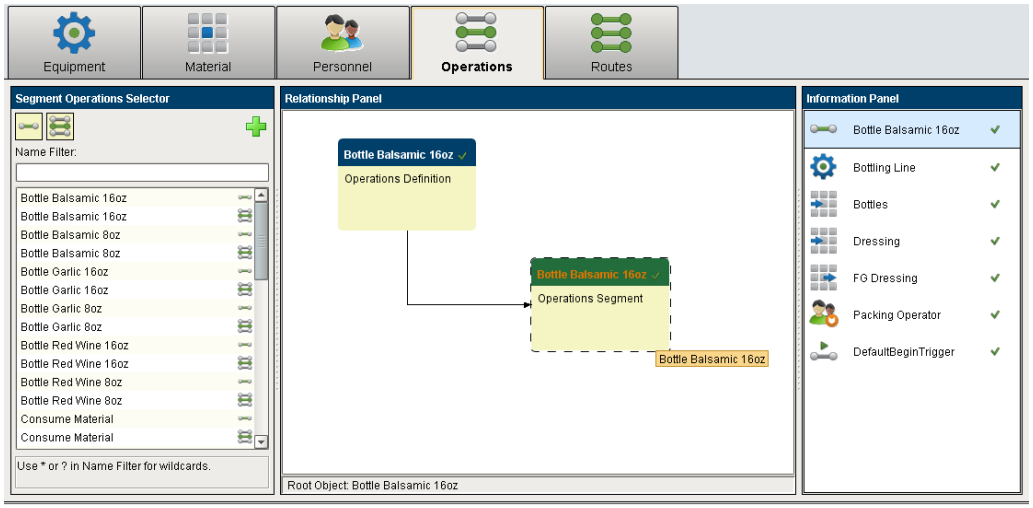


Operations Segment

Operations Segments are not created directly. Instead, they are derived from a Process Segment or an existing Operations Segment. This is typically done using the MES Object Editor component, but can also be done using script functions. When a new Operations Segment is derived from a Process Segment or Operations Segment, the core, material resource, personnel resource and equipment resource properties are copied over from the derived object. When updating a Process or Operations Segment using the MES Object Editor component, the user will be asked if they want to update the dependencies of any derived Operations Segments. If they answer yes, the changes will be pushed to all Process Segments or Operations Segments that were derived from the modified Process Segment.

This object inherits the AbstractMESObject properties and methods.

Goto Operations Segment section



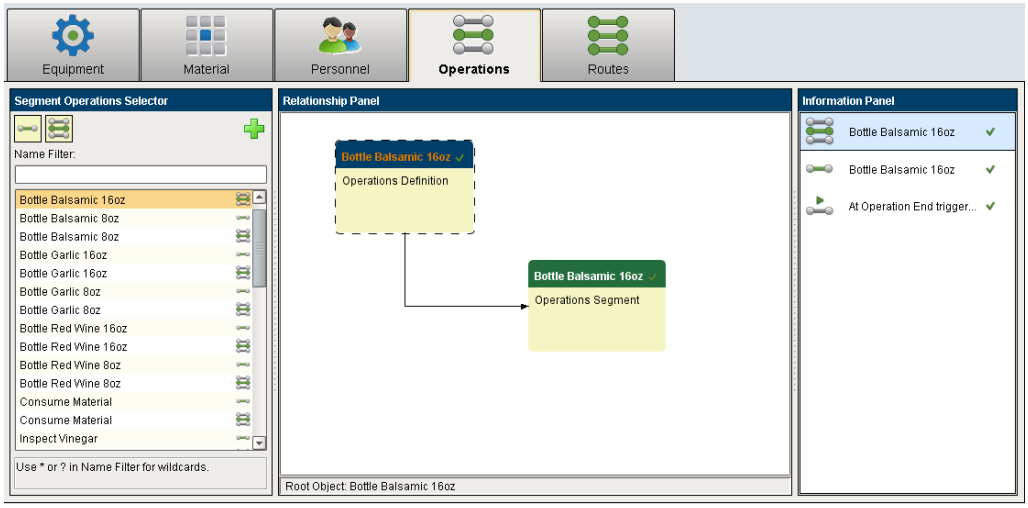
Operations Definition

The Operations Definition object is used to hold one or Operations Segments (basic tasks) to form a single Operation with multiple steps. Most commonly, there will be a one to one relationship of an Operations Definition i.e. 'Unload Material' having a single Operations Segment called 'Unload Material' which is derived from a Process Segment called 'Unload Material'.

In Track & Trace, Operations are selected, the segment within the operation is selected and executed and response objects are created containing the actual production results of that Operations Segment.

This object inherits the AbstractMESObject properties and methods and extends it with some custom properties.

Goto Operations Definition section



Operations Definition

Base Object

The Operations Definition is derived from the MESAbstractObject and inherits all the exposed properties, methods and events for that object.

Scripting Functions

The following scripting functions exists for a Operations Definition...

General Functions

There are many different types of MES objects in the Sepasoft MES system. All of them are inherited from the AbstractMESObject. Many of the scripting functions and properties refer to the commonAbstractMESObject objects. This page details the properties, functions and events that are common to all objects that are inherited from the AbstractMESObject.

Core Properties common to all MES Objects

Setting Name	Type	Description
name	read only	This is the name of the MES object. This name is used when referencing the object. It must be a unique name meaning that no other MES object of it's type can have the same name.
UUID	read only	This will contain the Universally Unique Identifier for each instance of a MES object.
enabled	write only	This property will be set to true when the MES object is active and usable. When MES objects are deleted they are still retained in the database and the Enabled setting is set to false. This is done to maintain past traceability information.
description	read-only	An optional settings to give more details for a MES Object.

Events

'New' Event

This event is run every time a new MES object is created. It can be used to add custom properties or to perform other tasks.

If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line:

Code Snippet

```
#Add custom property when a new instance of a MaterialDef object
is created.
obj = event.getMESObject()
obj.addCustomProperty('Width', 'Int4', 'Part Width', 'mm', True, F
alse)
event.runDefaultHandler()
```

Script Functions

The script functions listed below are available for all MES objects. They are used to simplify and reduce the number of lines of script for common tasks. An example is adding children, adding custom properties, changing property values, etc.

All of these script functions require an instance of a MES object. There are a number of methods to get an instance of an MES object and the code snippets below show just a couple of them.

Code Snippet

```
#Get the MES object for a given name and MES object type
obj = system.mes.loadMESObject('Balsamic Vinegar', 'MaterialDef')
```

Code Snippet

```
#If a link was returned from another script function, then this
will return the full MES object instance
obj = objLink.getMESObject()
```

Object Description

The Operations Definition object is used to hold one or more Operations Segments (basic tasks) to form a single Operation with multiple steps. Most commonly, there will be a one to one relationship of an Operations Definition i.e. 'Unload Material' having a single Operations Segment called 'Unload Material' which is derived from a Process Segment called 'Unload Material'.

In Track & Trace, Operations are selected, the segment within the operation is selected and executed and response objects are created containing the actual production results of that Operations Segment.

Events

Besides the common MES object events, the following events exist for MES Work Order.

Setting Name	Description
New	The event is fired when a new instance of an Operations Definition object is created. If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()

Extended Properties

Besides the common core properties , the following core properties exist for Operations Definition objects.

Setting Name	R/W	Description
Enable Update Event	Read /Write	When this setting is set to true, the UpdateProgress event for the Response Segments associated with this Operations Definition will be executed at the interval set in the Update Event Interval. The UpdateProgress event is defined in the Ignition Designer in the MES Events section.
MESTrackProgressByProperty	Read /Write	Production can be tracked by two factors namely, time and material in each segment. The default is to track by time and the option to track by material will only show if in the associated process segment material has been defined and a rate has been specified.
MESReadyToExecuteProperty	Read /Write	It is true only when there is a response segment to act upon. Operations Definition will not be ready for production unless all the segments are completely setup. And it is false if there is any error when validating operation definition object.
MESProductionVisibleProperty	Read /Write	If true, display operation in the selectors.
Update Event Interval	Read /Write	This setting defined the frequency (in ms) to execute the UpdateProgress event.

Segment Dependency Property

Segment Dependency properties are added as needed to define the Operations Segments that are associated with the operation. In the MES Object Editor component, the options shown when selecting a segment are Process Segments. When the Operations Definition is saved, Operations Segments will be derive from the Process Segment behind the scenes. Existing Operations Segments cannot be added because the equipment resource properties may not apply.

When creating a new Operations Definition as a child of an existing Operations Definition, new Operations Segments will be created for each of the parent's Operations Segments. The setting name is what appears in the MES Object Editor component and the script name is what is used to set or get the value using script. See AbstractMESComplexProperty for details about accessing values using script.

Setting Name	Script Name	Description
Segment Dependency Name	SegmentRef	This is the name to refer to this segment dependency resource by. Operations Definitions may have multiple segment resources and this is a unique name displayed to the operator, shown in analysis and reports, and also internally used to reference this segment dependency.
Segment Reference	SegmentRef	The reference to the Process Segment or Operations Segment that this segment dependency is link to.
Segment Dependency Type	SegmentRefType SegmentRefUUID	Currently this is here for reference and is included to align with the ISA-95 standard and will become significant in the next phase of the Track and Trace Module.
Segment Dependency Factor	SegmentDependencyFactor	Currently this is here for reference and is included to align with the ISA-95 standard and will become significant in the next phase of the Track and Trace Module
Segment Dependency Factor Units	SegmentDependencyFactorUnits	This is the units for the value of the Segment Dependency Factor setting.

Operations Segment

Base Object

The Operations Segment is derived from the MESAbstractObject and inherits all the exposed properties, methods and events for that object.

Scripting Functions

The following script functions exist for the Operations Segment.

getEquipment

Description

Return the equipment MES object associated with the segment. This will be the same equipment that is associated with the operation that the segment is running under.

Syntax

getEquipment()

- Parameters

None

- Returns

The AbstractMESObject representing the equipment.

- Scope

All

getEquipmentLink

Description

Return the link to the equipment MES object associated with the segment. This will be the same equipment that is associated with the operation that the segment is running under.

Syntax

getEquipmentLink()

- Parameters

None

- Returns

The MES Object Link representing the equipment.

- Scope

All

getEquipmentProperty

Description

Return the complex property of the equipment MES object associated with the segment. This will be the same equipment that is associated with the operation that the segment is running under.

Syntax

getEquipmentProperty()

- Parameters

None

- Returns

The MESEquipmentProperty representing the equipment.

- Scope

All

Object Description

Operations Segments are not created directly. Instead, they are derived from a Process Segment or an existing Operations Segment. This is typically done using the MES Object Editor component, but can also be done using script functions. When a new Operations Segment is derived from a Process Segment or Operations Segment, the core, material resource, personnel resource and equipment resource properties are copied over from the source object to the derived object. When updating a Process or Operations Segment using the

MES Object Editor component, the user will be asked if they want to update the dependencies of any derived Operations Segments. If they answer yes, the changes will be pushed to all Process Segments or Operations Segments that were derived from the modified Process Segment.

Events

Besides the common MES object events, no other events exist for the Operations Segment object.

Properties

Material Resource Property

Material resource properties are added as needed to define the materials feeding into or out of a Process Segment.

Setting Name	Description
Material Resource Name	This is the name to refer to this material resource by. Many process segments have multiple material resources and this is a unique name displayed to the operator, shown in analysis and reports, and also internally used to reference this material resource.
Material Reference	This can be set to a Material Class or a Material Definition. By setting this to a Material Class will cause the operator to be prompted for the specific material for this material resource. If set to a Material Definition, then the selection will be automatically selected.
MES Object Name	Depending on the type set, Material Class or Material Definition, options will show. When Process Segments or Operations Segments are inherited from a Process Segment, the options that show for the Material Reference setting will be limited to the settings in the parent . For example: If Vinegar Material class is selected for a Process Segment and a new child Process Segment is created from it, then the options will be limited to the Vinegar Material Class for any child of it.
Units	This appears to the operator, analysis and reports.
Use	This is a very important setting and complete understanding of the possible options is critical.

Setting Name	Description
	Options **In** - is used for material feeding into a segment that will be part of the finished goods. **Out** - is used for material feeding out of a segment that is or will be part of the finished goods. **Consumable** - is used for material feeding into a segment that is not part of the finished goods. **By-product** - is used for material feeding out of a segment that is not part of the finished goods.
Lot Number Source	This determines the source of the lot number. ***Options*** **Manual** - prompt the operator for the lot number. This is typically used when receiving raw materials or entering a lot number generated by an outside system. **Auto** - automatically generated lot number. The internal lot number generator will generate a lot number and assign it automatically for the operator. This option can also be used if a different lot number format is used or lot numbers are provided by another system that is integrated with this system. **In Link** - In cases where the lot number of output of a segment will be the same as the lot number of one of the inputs of the same segment, this setting will tie the two together. Segments can be configured with multiple material inputs and outputs and different lot number links can be configured.
Lot Number Source Link	If the Lot Number Source setting is set to In Link, then this is the name of the material resource to get the lot number from.
Auto Generate Lot	If true, a new lot will be generated for the material output of a segment. This is typically only done when receiving material and a lot doesn't currently exist.
	This is the type of equipment associated with this material resource. It can be set to a Material Class or a specific piece of equipment.

Setting Name	Description
Lot Equipment Reference	In the case where this material resource is an input to the segment, this will be the equipment where the material is coming from. This helps establish routes that exist for physical machinery. For example if tanks 1 and 2 can only supply line 1 and tanks 3 and 4 can only supply line 2. In the case where this material resource is an output from the segment, this will be the equipment where the material is going to. And as was the case for the input, routes that exist for physical machinery can be accommodated.
MES Object Name	Depending on the setting of the type, Equipment Class, Equipment, Line, Line Cell, Line Cell Group or Storage Unit options will show. When Process Segments or Operations Segments are inherited from a Process Segment, then the options that show for the Material Reference setting will be limited by the settings in the parent. For example: If Vinegar Tanks class is selected for a Process Segment and a new child Process Segment is created from it, then the only options will be limited to the Vinegar Tanks class and any child of it.
Quantity	Typically this is left blank, but it can be set to a fixed value that will be constant every time the segment is used for production.
Quantity Source	This setting determines the source of the quantity for this material resource. ***Options*** **Available Lot Quantity** - The available lot quantity for the incoming lot will be used for the Quantity Source. **Link** - This option allows the quantity to come from an input or output material resource of this segment. This eliminates the need to type in the quantity multiple times if they will always be the same as another material resource. **Link Combine** - For segments that are combining two or more lots into one stream, as is the case of joining goods after tests are done to only a portion of a lot, this option can be used. It is used by having two or more material resources that are segment inputs linked to the same material resource output. When the segment is ended, the system will add the quantities of both material resources. Delimiters such as "," or any other delimiter is unnecessary while defining the quantity source link, instead a single name is used that can be put into all of the material references link name field .

Setting Name	Description
	Link Split - For segments that are splitting a lot into two or more streams, as is the case of separating good product from bad, this option can be used. It is used by having two or more material resources, that are segment outputs, linked to the same material resource. When the segment is ended, the system will ensure that the sum of the quantities of the split material resources equals that of the linked material resources. **Manual** - The operator will be prompted for the quantity. The quantity must be entered before the segment is ended. **MES Counter** - Obtain the quantity from the automatic production counters defined for the associated equipment. The associated equipment may change if the Lot Equipment Reference setting is set to a Material Class and the specific equipment is not known until the segment is started for production. More information can be found in the MES Counters page. **Sublot Count** - The quantity will be automatically set based on the number of Material Sublot items belonging to the Material Lot. If sublots are used, then serial numbers, or other unique identification number, can be assigned to each sublot item. For example, a Material Lot of batteries may have 25 individual batteries each with a serial number and each with their own test result. The quantity of the Material Lot will match the number of Material Sublot items of the Material Lot. Or, in this case, the number of batteries in the lot.
Quantity Source Link	This is used when the Quantity Source setting is set to Link, Split or Combine. It is the name of the material resource to link to for this segment. On the **In** s, if Quantity Source is set to Available Lot Quantity or Manual, and the Quantity Source Link to "combine", then on the **Out** s, set the Quantity Source to "Link Combine" and the Quantity Source Link also to "combine".
Final Lot Status	When a segment is started, the status of the Material Lots will be set to Active. When the segment is ended or a new lot is used for the material resource, the status will be set to Complete. Optionally, the value of this setting can be used instead of the default Complete. Please note, the Active status while the lot is active cannot be changed. This is useful for setting a lot to Hold, In Process or anything that can be used to filter lots or sublots.

Setting Name	Description
Enable Sublots	If this setting is selected, then sublot support will be enabled for the material resource. If sublots are used, then serial numbers, or other unique identification number, can be assigned to each sublot item. For example, a Material Lot of batteries maybe have 25 individual batteries each with a serial number and each with their own test result.
Rate Period	This is used to set the material rate period. ***Options*** **Min** - For setting the rate in minutes. **Hour** - For setting the rate in hours. **Cycle** - For setting the rate in cycles.

Personnel Resource Property

Personnel resource properties are added as needed to define the people that are required for the Process Segment.

Setting Name	Description
Personnel Resource Name	This is the name to refer to this personnel resource by. Some process segments have multiple personnel resources and this is a unique name displayed to the operator, shown in analysis and reports, and also internally used to reference this personnel resource.
Personnel Reference Type	This can be set to a Personnel Class or a Person. By setting this to Personnel Class will cause the operator to be prompted for the specific Person for this personnel resource. If set to a Person, then the selection will be automatically selected.
MES Object Name	Depending on the setting of the type, Personnel Class or Person options will show. When Process Segments or Operations Segments are inherited from a Process Segment, then the options that show for the Personnel Reference setting will be limited by the parent settings.

Setting Name	Description
	For example: If Unload Operator Personnel class is selected for a Process Segment and a new child Process Segment is created from it, then the only options will be limited to the Unload Operator Class and any child of it.
Units	This specifies the units for the quantity setting.
Use	Currently this is here for reference and is included to align with the ISA-95 standard and will become significant in the next phase of the Track and Trace Module.
Quantity	Currently this is here for reference and is included to align with the ISA-95 standard and will become significant in the next phase of the Track and Trace Module.

Equipment Resource Property

An equipment resource property is added to define the equipment that the Process Segment will run on.

Setting Name	Description
Equipment Resource Name	This is the name to refer to this equipment resource by. Some process segments have multiple equipment resources and this is a unique name displayed to the operator, shown in analysis and reports, and also internally used to reference this equipment resource.
Equipment Reference	This can be set to a Equipment Class or a Equipment, Line, Line Cell, Line Cell Group or Storage Unit. By setting this to Equipment Class will cause the operator to be prompted for the specific equipment for this equipment resource. If set to a specific equipment item, then the selection will be automatically selected.
MES Object Name	Depending on the setting of the type, Material Class or specific equipment item options will show. When Process Segments or Operations Segments are inherited from a Process Segment, then the options that show for the Equipment Reference setting will be limited by the parent settings.

Setting Name	Description
	For example: If Unload Stations class is selected for a Process Segment and a new child Process Segment is created from it, then the only options will be limited to the Unload Stations Class and any child of it.
Units	This specifies the units for the quantity setting.
Use	Currently this is here for reference and is included to align with the ISA-95 standard and will become significant in the next phase of the Track and Trace Module.
Quantity	Currently this is here for reference and is included to align with the ISA-95 standard and will become significant in the next phase of the Track and Trace Module.

Process Segment

Base Object

The Process Segment is derived from the MESAbstractObject and inherits all the exposed properties, methods and events for that object.

Scripting Functions

The following script functions exist for the Process Segment.

getEquipment

Description

Return the equipment MES object associated with the segment. This will be the same equipment that is associated with the operation that the segment is running under.

Syntax

getEquipment()

- Parameters

None

- Returns

The AbstractMESObject representing the equipment.

- Scope

All

getEquipmentLink

Description

Return the link to the equipment MES object associated with the segment. This will be the same equipment that is associated with the operation that the segment is running under.

Syntax

getEquipmentLink()

- Parameters

None

- Returns

The MES Object Link representing the equipment.

- Scope

All

getEquipmentProperty

Description

Return the complex property of the equipment MES object associated with the segment. This will be the same equipment that is associated with the operation that the segment is running under.

Syntax

getEquipmentProperty()

- Parameters

None

- Returns

The MESEquipmentProperty representing the equipment.

- Scope

All

Object Description

The Process Segment is an object used to hold the definition of basic tasks that are performed in a manufacturing facility. Tasks can be as simple as 'Produce Product' or 'Unload Material', but can be more granular like 'Changeover Line', 'Clean', 'Lab Inspection', 'Heat Up', 'Pre-Production', 'Produce Product' and 'Run Out Line'.

The Process Segment is never used in an actual operation, but acts as a template for Operation Segments to be created and based against.

Events

Besides the common MES object events, the following events exist for Process Segment.

Setting Name	Description
New	The event is fired when a new instance of an Process Segment object is created. If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()

Properties

Material Resource Property

Material resource properties are added as needed to define the materials feeding into or out of a Process Segment. The setting name is what appears in the MES Object Editor component and the script name is what is used to set or get the value using script. See AbstractMESComplexProperty for details about accessing values using script.

Setting Name	Script Name	Description
Material Name	MaterialName	This is the name to refer to this material resource by. Many process segments have multiple material resources and this is a unique name displayed to the operator, shown in analysis and reports, and also internally used to reference this material resource.
Material Reference	MaterialRef	This can be set to a Material Class or a Material Definition. By setting this to Material Class will cause the operator to be prompted for the specific material for this material resource. If set to a Material Definition, then the selection will be automatically selected.
MES Object Name		Depending on the setting of the type, Material Class or Material Definition options will show. When Process Segments or Operations Segments are inherited from a Process Segment, then the options that show for the Material Reference setting will be limited by the parent settings. For example: If Vinegar Material class is selected for a Process Segment and a new child Process Segment is created from it, then the only options will be limited to the Vinegar Material Class and any child of it.
	MaterialRefUUID	UUID of the Material Class or Material Definition.
Units	MaterialUnits	This appears to the operator, analysis and reports.
Use	MaterialUse	Material use property. Options are: **In -** This setting is used for material feeding from an existing material lot into a segment that will be part of the finished goods.

Setting Name	Script Name	Description
		Out - This setting is used for material feeding out of a segment that is or will be part of the finished goods. **Consumable -** This setting is used for material feeding into a segment that is not part of the finished goods. **By-product -** This is used for material feeding out of a segment that is not part of the finished goods.
Lot Number Source	MaterialLotNoSource	This determines the source of the lot number. ***Options*** **Manual** - prompt the operator for the lot number. This is typically used when receiving raw materials or entering a lot number generated by an outside system. **Auto** - automatically generated lot number. The internal lot number generator will generate a lot number and assign it automatically for the operator. This option can also be used if a different lot number format is used or lot numbers are provided by another system that is integrated with this system. **In Link** - In cases where the lot number of output of a segment will be the same as the lot number of one of the inputs of the same segment, this setting will tie the two together. Segments can be configured with multiple material inputs and outputs and different lot number links can be configured.
Lot Number Source Link	MaterialLotNoSourceLink	If the Lot Number Source setting is set to In Link, then this is the name of the material resource to get the lot number from.

Setting Name	Script Name	Description
Auto Generate Lot	MaterialAutoGenerateLot	If true, a new lot will be generated for the material output of a segment. This is typically only done when receiving material that a lot doesn't already exists.
Lot Equipment Reference	MaterialEquipmentRef	This is the type of equipment associated with this material resource. It can be set to a Material Class or a specific piece of equipment. In the case where this material resource is an input to the segment, this will be the equipment where the material is coming from. This helps establish routes that exist due to physical machinery. For example if tanks 1 and 2 can only supply line 1 and tanks 3 and 4 can only supply line 2. In the case where this material resource is an output from the segment, this will be the equipment where the material is going to. And just like the case of the input, routes that exist due to physical machinery can be accommodated.
MES Object Name		Depending on the setting of the type, Equipment Class, Equipment, Line, Line Cell, Line Cell Group or Storage Unit options will show. When Process Segments or Operations Segments are inherited from a Process Segment, then the options that show for the Material Reference setting will be limited by the parent settings. For example: If Vinegar Tanks class is selected for a Process Segment and a new child Process Segment is created from it, then the only options will be limited to the Vinegar Tanks class and any child of it.
	MaterialEquipmentRefUUID	UUID for the Equipment Class or Equipment item.

Setting Name	Script Name	Description
Quantity	MaterialQuantity	Typically this is left blank, but it can be set to a fixed value that will be constant every time the segment is used for production.
Quantity Source	MaterialQuantitySource	This setting determines the source of the quantity for this material resource. ***Options*** **Available Lot Quantity** - Number of items belonging to the lot can be obtained with this feature. **Link** - This option allows the quantity to come from an input or output material resource of this segment. This eliminates the need to type in the quantity multiple times if they will always be the same as another material resource. **Link Combine** - For segments that are combining two or more lots into one streams, as is the case of joining goods after tests are done to only a portion of a lot, this option can be used. It is used by having two or more material resources, that are segment inputs, linked to the same material resource output. When the segment is ended, the system will sum up the quantities of the linked material resources to that of the linking material resources. There isn't a need to use "," or any other delimiters while defining the quantity source link, instead a single name is used that can be put into all of the material references link names. **Link Split** - For segments that are splitting a lot into two or more streams, as is the case of separating good from bad product, this option can be used. It is used by having two or more material resources, that are segment outputs, linked to the same material resource. When the segment is

Setting Name	Script Name	Description
		ended, the system will ensure that the sum of the quantities of the linking material resources equal that of the linked material resources. **Manual** - The operator will be prompted for the quantity. The quantity must be entered before the segment is ended. **MES Counter** - Obtain the quantity from the automatic production counters defined for the associated equipment. The associated equipment may change if the Lot Equipment Reference setting is set to a Material Class and the specific equipment is not known until the segment is started for production. More information can be found in the MES Counters page. **Sublot Count** - The quantity will be automatically set based on the number of Material Sublot items belonging to the Material Lot. If sublots are used, then serial numbers, or other unique identification number, can be assigned to each sublot item. For example, a Material Lot of batteries maybe have 25 individual batteries each with a serial number and each with their own test results. The quantity of the Material Lot will match the number of Material Sublot items of the Material Lot. Or, the number of batteries in the lot.
Quantity Source Link	MaterialQuantitySourceLink	This is used when the Quantity Source setting is set to Link, Split or Combine. It is the name of the material resource to link to this segment. On the **In**s, if Quantity Source is set to Available Lot Quantity or Manual, and the Quantity Source Link to "combine", then on the **Out**s, set the Quantity Source to "Link Combine" and the Quantity Source Link also to "combine".
	MaterialFinalLotStatus	

Setting Name	Script Name	Description
Final Lot Status		When a segment is started, the status of the Material Lots will be set to Active. When the segment is ended or a new lot is used for the material resource, the status will be set to Complete. Optionally, the value of this setting can be used instead of the default Complete. Please note, the Active status while the lot is active cannot be changed. This is useful for setting a lot to Hold, In Process or anything that can be used to filter lots or sublots.
Enable Sublots	MaterialEnableSublots	If this setting is selected, then sublot support will be enabled for the material resource. If sublots are used, then serial numbers, or other unique identification number, can be assigned to each sublot item. For example, a Material Lot of batteries may have 25 individual batteries each with a serial number and each with their own test results.
Rate Period	MaterialRatePeriod	This is used to set the material rate period. ***Options*** **Min** - For setting the rate in minutes. **Hour** - For setting the rate in hours. **Cycle** - For setting the rate in cycles.

Personnel Resource Property

Personnel resource properties are added as needed to define the people that are required for the Process Segment. The setting name is what appears in the MES Object Editor component and the script name is what is used to set or get the value using script. See AbstractMESComplexProperty for details about accessing values using script.

Setting Name	Script Name	Description
Personnel Name		This is the name to refer to this personnel resource by. Some process segments have multiple personnel resources and this is a unique name displayed to the operator, shown in analysis and reports, and also internally used to reference this personnel resource.
Personnel Reference	PersonnelRef	This can be set to a Personnel Class or a Person. By setting this to Personnel Class will cause the operator to be prompted for the specific Person for this personnel resource. If set to a Person, then the selection will be automatically selected.
MES Object Name		Depending on the setting of the type, Personnel Class or Person options will show. When Process Segments or Operations Segments are inherited from a Process Segment, then the options that show for the Personnel Reference setting will be limited by the parent settings. For example: If Unload Operator Personnel class is selected for a Process Segment and a new child Process Segment is created from it, then the only options will be limited to the Unload Operator Class and any child of it.
	PersonnelRefUUID	UUID of the selected Personnel Class or Person.
Units	PersonnelUnits	This specifies the units for the quantity setting.
Use	PersonnelUse	Currently this is here for reference and is included to align with the ISA-95 standard and will become significant in the next phase of the Track and Trace Module.
Quantity	PersonnelQuanity	Currently this is here for reference and is included to align with the ISA-95 standard and will become significant in the next phase of the Track and Trace Module.

Equipment Resource Property

An equipment resource property is added to define the equipment that the Process Segment will run on. The setting name is what appears in the MES Object Editor component and the script name is what is used to set or get the value using script. See AbstractMESComplexProperty for details about accessing values using script.

Setting Name	Script Name	Description
Equipment Resource Name		This is the name to refer to this equipment resource by. Some process segments may have multiple equipment resources and this is a unique name displayed to the operator, shown in analysis and reports, and also internally used to reference this equipment resource.
Equipment Reference	EquipmentRef	This can be set to a Equipment Class or a Equipment, Line, Line Cell, Line Cell Group or Storage Unit. By setting this to Equipment Class will cause the operator to be prompted for the specific equipment for this equipment resource. If set to a specific equipment item, then the selection will be automatically selected.
MES Object Name		Depending on the setting of the type, Equipment Class or specific equipment item options will show. When Process Segments or Operations Segments are inherited from a Process Segment, then the options that show for the Equipment Reference setting will by limited by the parent settings. For example: If Unload Stations class is selected for a Process Segment and a new child Process Segment is created from it, then the only options will be limited to the Unload Stations Class and any child of it.
	EquipmentRefUUID	UUID of the Equipment Class or Equipment item.
Units	EquipmentUnits	This specifies the units for the quantity setting.
Use	EquipmentUse	Currently this is here for reference and is included to align with the ISA-95 standard and will become significant in the next phase of the Track and Trace Module.
Quantity	EquipmentQuantity	

Setting Name	Script Name	Description
		Currently this is here for reference and is included to align with the ISA-95 standard and will become significant in the next phase of the Track and Trace Module.

Equipment Objects

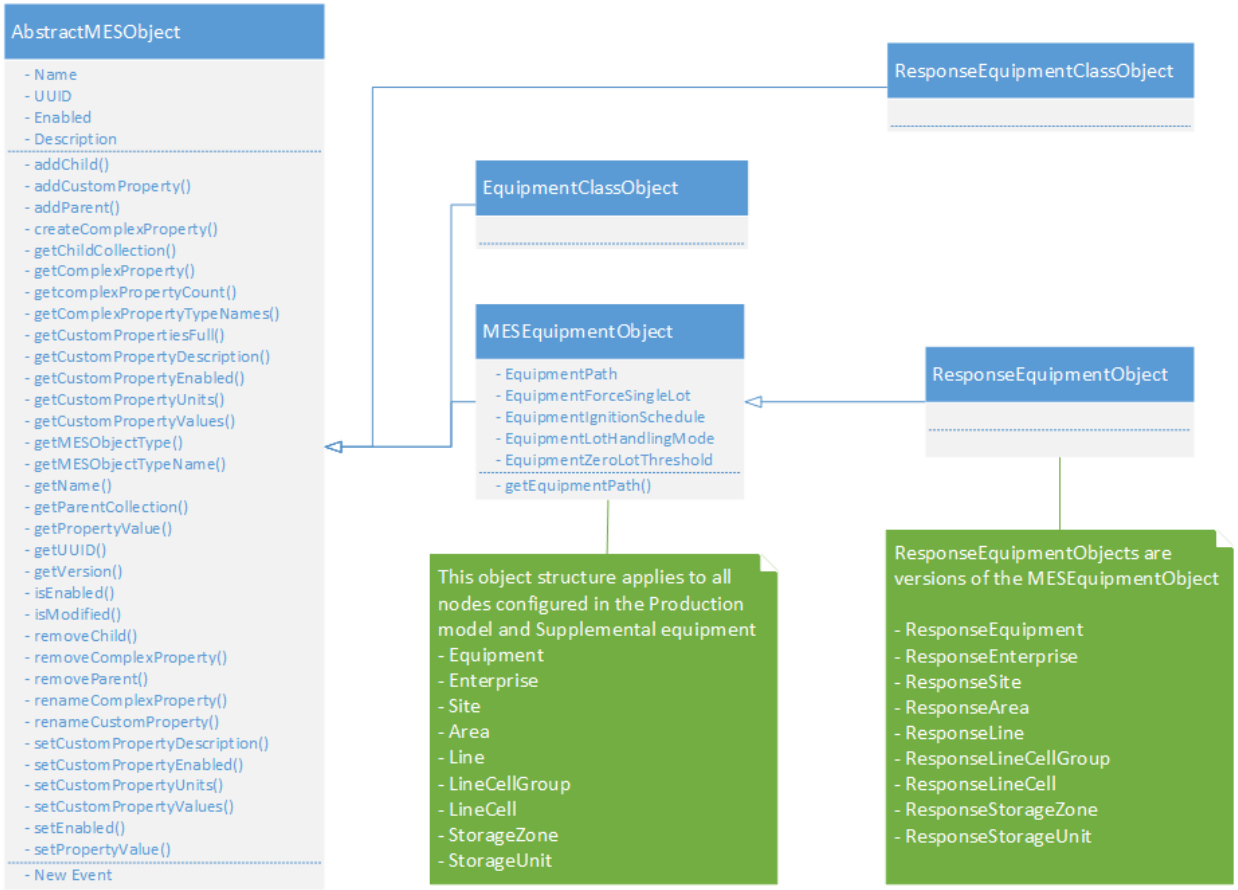
Equipment MES objects are created when the production model is started. Only one Equipment MES object is created for every production item in the production model that is defined in the designer. Multiple Equipment class MES objects can be created in the MES object Editor component and equipment assigned to them.

There are two main Object Types in the Equipment Objects Group, **MESEquipmentObject** and **MESEquipmentClassObject**.

Both these objects have a corresponding **ResponseEquipmentObject** and **ResponseEquipmentClassObject** which is an internal versioning schema created to maintain historical production data whenever changes are made to the MESEquipment or EquipmentClass objects.

The **MESEquipmentObject** and **MESEquipmentClassObject** inherit the AbstractMESObject properties and methods and also extend it with the same properties and methods as shown further down in this section.

The **MESEquipmentClassObject** and **ResponseEquipmentClassObject** inherit the AbstractMESObject properties and methods but do not extend it in anyway.



Equipment Object Versions

Every time an Equipment Object is modified, i.e. adding custom properties, changing a setting etc., the version number of that equipment object will be updated in the background. When an operation is scheduled, it will check for a corresponding ResponseObject version. If one does not exist, it will automatically create a new Response object.

This versioning is not part of ISA-95, however, without it, analysis of historical data would lose the original configuration of equipment, personnel and materials.

For all intents and purposes, MES equipment objects will be created and configured in the Production model, from the MES Management screen and through scripting. Response Objects are automatically created by Operations and will be used for any kind of traceability analysis.

Although these are called **Response** Objects, they are in fact **Version** objects of the **Equipment** Objects. They are not Response Segment objects as defined by ISA-95.

MESEquipment and ResponseEquipment Objects

With the exception of the 'Equipment' and 'ResponseEquipment' objects, all the **MESEquipment** and **ResponseEquipment** Objects form a hierarchy that follows that of the Production model (from Enterprise, Site, Area, Storage Zone, Storage Unit, Line, and CellGroup down to the Cell), and these are created automatically on startup of the MES system.

Supplemental Equipment

The **Equipment** and **ResponseEquipment objects** are used to define supplemental equipment (rolling or mobile such as bins, pallets) that can be included for a segment. This equipment does not reside in the production model and operations cannot be performed on them.

The following table summarizes the different Equipment Objects that inherit from the MESEquipmentObject and what is possible.

Equipment Object	Response Object (Version)	Description	Auto Created	Can Run Operation	Can Perform Analysis
Equipment	ResponseEquipment	Supplemental equipment only			✓
Enterprise	ResponseEnterprise	Object based on the Enterprise configured in the production model.	✓		
Site	ResponseSite	Object based on the Site configured in the production model.	✓		✓
Line	ResponseLine		✓	✓	✓

Equipment Object	Response Object (Version)	Description	Auto Created	Can Run Operation	Can Perform Analysis
		Object based on the Line configured in the production model.			
LineCellGroup	ResponseLineCellGroup	Object based on the LineCellGroup configured in the production model.	✅		✅
LineCell	ResponseLineCell	Object based on the LineCell configured in the production model.	✅	✅	✅
StorageZone	ResponseStorageZone	Object based on the StorageZone configured in the production model.	✅		✅
StorageUnit	ResponseStorageUnit	Object based on the StorageUnit configured in the production model.	✅	✅	

Extended Script Functions

The following script functions exist for the Equipment and ResponseEquipment objects only.

Extended Functions

getEquipmentPath()

Description

Gets the path associated with this equipment object.

Syntax

getEquipmentPath()

- Parameters

None

- Returns

String EquipmentPath - The path associated with this equipment.

- Scope

All

getEquipmentModeClassUUID()

Description

Gets uuid of the equipment mode class.

Syntax

getEquipmentModeClassUUID()

- Parameters

None

- Returns

String uuid - The uuid that represents the equipment mode class.

- Scope

All

getEquipmentStateClassUUID()

Description

Gets uuid of the equipment state class.

Syntax

getEquipmentStateClassUUID()

- Parameters

None

- Returns

String uuid - The uuid that represents the equipment state class.

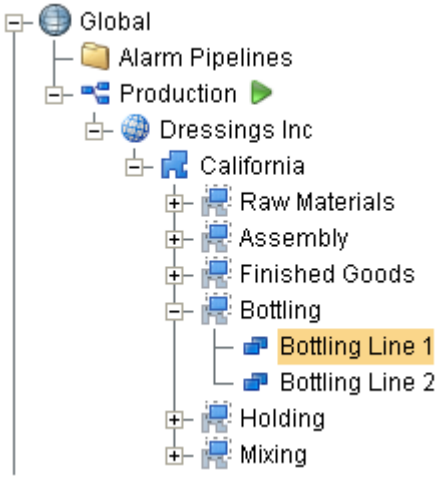
- Scope

All

Extended Properties

Besides the common core properties, the following core properties exist for Equipment and ResponseEquipment objects only.

Setting Name	Description
EquipmentPath	Equipment path is the location path for the specified equipment.

Setting Name	Description
	The equipment path is based on the hierarchy in the item in the production model. Referring to the image below, the equipment path for Bottling Line 1, is "Dressings Inc\California\Bottling\Bottling Line 1". Equipment paths are used by some of the MES components and script functions to easily identify equipment items. 
EquipmentForceSingleLot	Force storage of a single lot. There should be at least one lot.
EquipmentIgnitionSchedule	Scheduling the equipment in Ignition.
EquipmentLotHandlingMode	This would handle the equipment lots. The different types of mode available are single lot, random lot, FIFO, LIFO, same lot, blend lot and unknown.
EquipmentZeroLotThreshold	If the lot threshold is less than 0, it returns Invalid lot.

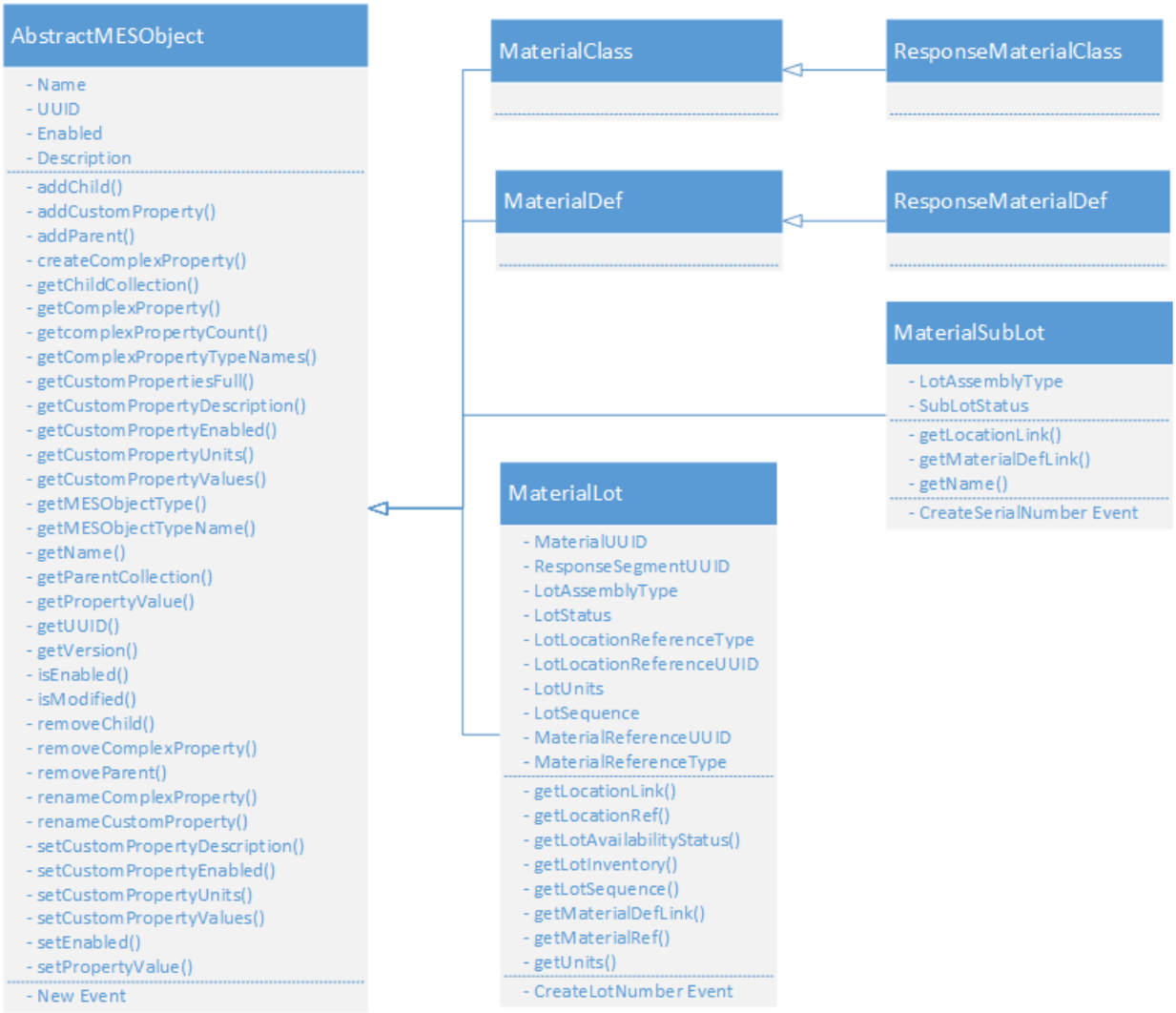
Material Objects

Any production or processing that is done involves material. The material maybe raw material that goes into finished goods, or it can be consumable or by-product that is not directly related to the finished good.

There are four Object Types in the Material Objects Group, **MaterialClass** and **MaterialDef**, **MaterialLot** and **MaterialSubLot**.

MaterialClass and **MaterialDef** objects have a corresponding **ResponseMaterialClass and ResponseMaterialDef,** which is an internal versioning schema created to maintain historical production data whenever changes are made to the properties or settings of the MaterialClass and MaterialDef objects.

All of these objects inherit the AbstractMESObject properties and methods. The **MaterialLot** and **MaterialSubLot** further extend the parent object with the properties and methods found further down in this section.



Material Object Versions

Every time a MaterialDef or Class Object is modified, i.e. adding custom properties, changing a setting etc., the version number of that equipment object will be updated in the background. When an operation is scheduled, it will check for a corresponding Response Object version. If one does not exist, it will automatically create a new Response object.

This versioning is not part of ISA-95, however, without it, analysis of historical data would lose the original configuration of equipment, personnel and materials.

For all intents and purposes, MES Material objects will be created and configured in the MES Management screen and through scripting. Response Objects are automatically created by Operations and will be used for any kind of traceability analysis.

Although these are called **Response** Objects, they are in fact **Version** objects of the Material Objects. They are not Response Segment objects as defined by ISA-95

Material Class

The MaterialClass object is used to group material into a category. It can have MaterialDef or another MaterialClass objects as children. Defining production tasks for each specific material, is very tedious. A better method would be to organize the material into categories, or class using ISA-95 terms. An example will make this clearer with fewer words. Consider unloading electronic components at a receiving dock. Defining a task to receive each type of component would be a management nightmare. Instead all of the components can be added to an Electronic Component class and when the operator does the receive components task at the dock, it prompts them for the specific component that belongs to the Electronic Components class. Only one receive components task has to be defined, which is much easier to manage.

The MaterialClass object inherits the AbstractMESObject properties and methods, but does not extend them.

Material Def

An object used to define material that can have Material Definition objects as children. Material definitions are used to define raw materials, material that are partially processed but not in finished goods state and finished goods. Consider the following case: If we are assembling an electronic product, then we will have electronic components, including a circuit board, that will each have material definitions. The components will be soldered to the circuit board and will have a material definition for the sub assembly. Next, the circuit board will be added to the housing which will have a material definition that represents it. This will continue until the finished goods are complete. It may even include accessories that are sold with the finished product. Each will have a material definition. Think of it this way: in order to know which lots of components were used to make a batch of circuit boards material definitions are needed.

The MaterialDef object inherits the AbstractMESObject properties and methods, but does not extend them.

Material Lot

Base Object

The Material Lot Object is derived from the MESAbstractObject and inherits all the exposed properties, methods and events for that object.

Scripting Functions

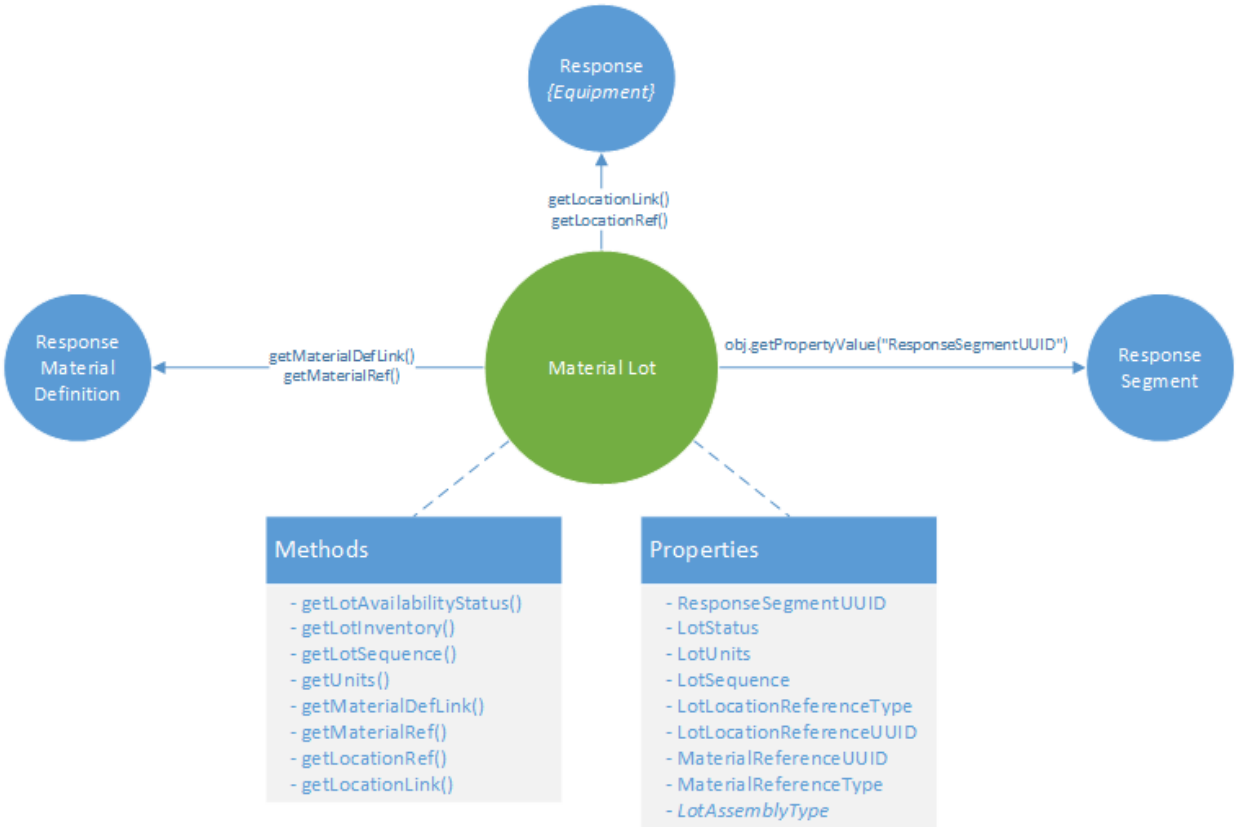
The following scripting functions return a Material Lot Object...

system.mes.createMESObject

system.mes.loadMaterialLot

Object Description

Material Lot Objects are used and created whenever an Operations Segment is executed. This object holds information about the Material Lot and can have Material Sublot objects as children. The Material Lot object does not store a great deal of information about itself, but stores references and links to other objects associated with it. Object Methods are provided that allow you to access the Response Segment that created the Material Lot, the Response Material Definition for the Material Lot and the Response *Equipment* (i.e. Response Storage Unit), where the Material Lot is located.



Material Sublots

When a segment is started, a Material Lot object is created for the OUT material. If the parts being produced by that segment are serialized, a Material Sublot object can be created for each serialized part that are associated with the Material Lot. The Material Sublots can be traced back throughout the process and show up as a highlight on the Material Lot object on the Trace Graph View. Creating Material Sublots is much more efficient than executing a process segment for each part produced. However re-introducing sublots back into a manufacturing process for re-work provides some challenges for traceability.

Example

```
obj = system.mes.loadMESObject('2dce886c-8ce6-4aeb-b271-
62adc07a6f26')    #Return a Material lot object
obj.getChildCollection().getList()
```

Material Lot Quantities

Material Lot Objects do not hold the quantity of material in that lot. The quantity consumed or created is kept in the **Lot Resource** properties of the **Response Segment** object which provides a mechanism for determining how much of a material lot was consumed or created by a given Response Segment.

In order to find out how much material remains in a lot, the material lot object method **obj.getLotInventory().getNetQuantity()** queries each of the response segments that have touched the Material Lot object to return the remaining lot quantity

References versus Links

The Material Lot object provides two methods to obtain the associated **Response Material Definition** and **Response *{Equipment}*** objects.

getLocationLink() and **getMaterialDefLink()** both return a lightweight object that can then be used to access the actual object i.e. **getLocationLink().getMESObject()** whereas **getLocationRef()** and **getMaterialRef()** return an AbstractMESObjectReferenceProperty object.

It is recommended to always use the ...Link() method as this provides a faster way of getting to the object directly, however, be advised that ...link().getMESObject() will throw the error *Information is missing for response material location reference* if either the UUID or Type is missing for the returned MESObject. getLocationRef() will not throw an error and allows you to access an object that has only one of the UUID or Type parameters set. This error generally only occurs if deriveMESObject() or createMESObject() was used to create an MES object without setting up all the necessary parameters. Using process segments to create the MES Objects will ensure that the objects have the parameters correctly setup.

Methods versus Properties

The Material Lot Object provides a set of methods that can be called to return object properties such as **obj.getUnits()** or **obj.getLotAvailabilityStatus()**. These method calls are recommended over directly accessing object properties using the obj.getPropertyValue ("LotStatus"). The reason for this is because of subtle differences between the value returned by the method and the property.

As an example, **obj.getLotAvailabilityStatus()** returns the enumerated type **AVAILABLE** whereas **myobj.getPropertyValue('LotAvailabilityStatus')** returns the string **Available.**

It is recommended to only access properties when no method currently exists to obtain the value of the property.

Methods

Beside the common MESAbstractObject methods, the following methods exist for the MaterialLot object.

getLocationLink

Description

Get the link to the location of the material lot.

Syntax

getLocationLink()

- Parameters

None

- Returns

The MES Object Link representing the location.

- Scope

All

getLocationRef

Description

Returns the location reference of the core property of material lot.

Syntax

getLocationRef()

- Parameters

None

- Returns

AbstractMESObjectReferenceProperty - The location reference of the material lot property.

- Scope

All

getLotAvailabilityStatus

Description

Returns the availability of the lot.

Syntax

getLotAvailabilityStatus()

- Parameters

None

- Returns

Availability status is returned.

- Scope

All

getLotInventory

Description

Get inventory of the material lot.

Syntax

getLotInventory()

- Parameters

None

- Returns

The MES Lot Quantity Summary Item object with inventory details.

- Scope

All

getLotSequence

Description

Returns the sequence number of the corresponding material lot.

Syntax

getLotSequence()

- Parameters

None

- Returns

Integer lotSequence - The sequence number associated with this material lot.

- Scope

All

getMaterialDefLink

Description

Get the link to the definition of the material lot.

Syntax

getMaterialDefLink()

- Parameters

None

- Returns

The MES Object Link representing the definition.

- Scope

All

getMaterialRef

Description

Returns the reference of the core property of material lot.

Syntax

getMaterialRef()

- Parameters

None

- Returns

AbstractMESObjectReferenceProperty - The reference of the material lot property.

- Scope

All

getUnits

Description

Gets the unit of corresponding material lot quantity.

Syntax

getUnits()

- Parameters

None

- Returns

String units - The units for the lot quantity.

- Scope

All

setLotSequence(lotSequence)

Description

Sets the sequence number for material lot.

Syntax

setLotSequence(lotSequence)

- Parameters

Integer lotSequence - The sequence number associated with this material lot.

- Returns

Nothing

- Scope

All

setUnits(lotUnits)

Description

Sets the unit of corresponding material lot quantity.

Syntax

setUnits(lotUnits)

- Parameters

String lotUnits - The units to set for the lot quantity.

- Returns

Nothing

- Scope

All

Events

Besides the common MES object events, the following events exist for Material Lot objects.

Setting Name	Description
CreateLotNumber	This event is run every time a Material Lot object is requested to create a new lot number. This event provides a method to intercept the generation of lot numbers so that the format of the lot number or even the number itself can be modified. For systems that retrieve lot numbers from a ERP or other system, this event will allow obtaining a lot number from it. Script in this event can also read from a block of available lot numbers that are maintained in a database.

Setting Name	Description
EvaluateLotStatus	The event is fired when a Material Lot is being finalized in a segment operation with a quantity and/or status change. If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()
New	The event is fired when a new instance of a Material Lot MES object is created. If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()

Properties

Property values can be accessed and changed for an object by using the getPropertyValue() and setPropertyValue() method.

Example

```
obj = system.mes.loadMESObject('2dce886c-8ce6-4aeb-b271-
62adc07a6f26')    #Return a Material lot object
print obj.getPropertyValue('LotStatus')
```

Besides the common core properties, the following properties exist for Material Lot objects.

Setting Name	R/W	Description
MaterialRefUUID	Read	The UUID of the Material Definition assigned to the Material Lot object.
MaterialRefType	Read	The type of the Material Definition object assigned to the Material Lot object. i.e. **ResponseMaterialDef**
ResponseSegmentUUID	Read	The UUID of the Response Segment that created the Material Lot object.

Setting Name	R/W	Description
LotAssemblyType	Read	Currently this is here for reference and is included to align with the ISA-95 standard and will become significant in the next phase of the Track and Trace Module.
Lot AvailabilityStatus	Read /Write	Returns the availability status of the lot. Default is Available. **Available** - Material Lot is currently available **Used** - Material Lot has been used up
LotStatus	Read /Write	The status of the Material Lot object. When a Material Lot object is currently being used by a Response Segment, this will be set to **Active**. When the Response Segment is ended, this will be set to the **Final Lot Status** setting in the Operation Segment configuration. If a Final Lot Status is not defined, it will be set to the default status of **Complete**.
LotLocationRefType	Read /Write	This is the type of equipment object where the lot is located i.e. **ResponseStorageUnit**
LotLocationRef UUID	Read	This is the UUID for the specific equipment of where the lot is located i.e. **6d15732b-5d64-41d7-a596-9ccef8eead15**
LotUnits	Read /Write	The unit defined for the lot quantity. Notice, that this is not a quantity setting for the Material Lot object. This is because the quantities are kept in the lot resource properties of the Response Segment object. The reason for this is that a many Response Segments may pull product from a single lot and it will have to be continually updated. In addition, details of how much a given Response Segment used of a lot still has to be maintained and storing quantities in the Material Lot object will be redundant which is never good.
LotSequence	Read /Write	

Setting Name	R/W	Description
		The sequence property is incremented every time a new Material Lot is created for a given lot number. This makes it unique as an unchanging lot number flows through a manufacturing facility.

Material Sublot

Object Description

When an operations segment is started, a Material Lot object is created for the OUT material. If the parts being produced by that segment are serialized, a Material Sublot object can be created for each serialized part that are associated with the Material Lot. The Material Sublots can be traced back throughout the process and show up as a highlight on the Material Lot object on the Trace Graph View. Creating Material Sublots is much more efficient than executing a process segment for each part produced. However re-introducing sublots back into a manufacturing process for re-work provides some challenges for traceability.

Creating Material Sublots

Material Sub lots can be created using the MES Sublot List component and through scripting functions exposed by the Response Segment.

Response Segment Sublot Functions

addSublot

Description

These script functions are used to add sublots one at a time to active segments and the different versions provide various methods to do so. Sublots are represented by MESMaterialSublot objects which corresponds to the ISA-95 Material Sublot objects. MESMaterialSublot objects must be children of a MESMaterialLot object, event though the lot information may not be needed. Usually, when production details are maintained for serialized items moving through production as groups, sublots are used. In cases where each item moves independently through production, then just material lots can be used for each serialized item.

Notice

In order for material sublots to be added, the Enable Sublots setting of the segment material property must be set to true.

addSublot(materialPropertyName, sublotName)

Description

This version of the addSublot script function is used to create a single new material sub lot. When the new is created, the CreateSerialNumber event of the MaterialSublot object will be executed and a serial number will automatically be assigned. This serial number, which is the name of the MaterialSublot object, can be changed prior updating the segment.

Syntax

addSublot(materialPropertyName, sublotName)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

String sublotName - The name of the sublot which is usually the serial number of the item.

- Returns

MESMaterialSublot - A new MESMaterialSublot object.

- Scope

All

Code Examples

Code Snippet

```
#Create a new segment
seg = system.mes.createSegment('Load Assembly Tray', '[globa
l]\Dressings Inc\California\Assembly\PS Assembly', False)
#Assign the material resources
seg.setMaterial('Housing', 'Housing', 'Assembly Tray 8')
```

```
#Begin the segment
seg.begin()
#Because the reference to the segment is different than the
one at the Ignition gateway after begin was executed,
refresh it.
seg = system.mes.getActiveSegment('[global]\Dressings
Inc\California\Assembly\PS Assembly', 'Load Assembly Tray')
#Create new sublots
seg.addSublot('Housing', 'SN 1234')
seg.addSublot('Housing', 'SN 2345')
seg.update()
```

addSublot(materialPropertyName, sublotName, customProperties)

Description

This version of the addSublot script function functions the same as the addSublot (materialPropertyName, sublotName) script function above with the added support to assign new custom properties to the new material sublot at the same time.

Syntax

addSublot(materialPropertyName, sublotName, customProperties)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

String sublotName - The name of the sublot which is usually the serial number of the item.

PyDictionary customProperties - A PyDictionary containing either name value pairs or complete custom property definitions. See Custom Properties for more information.

- Returns

MESMaterialSublot - A new MESMaterialSublot object.

- Scope

All

Code Examples

Code Snippet

```
#Create a new segment
seg = system.mes.createSegment('Load Assembly Tray', '[globa
l]\Dressings Inc\California\Assembly\PS Assembly', False)
#Assign the material resources
seg.setMaterial('Housing', 'Housing', 'Assembly Tray 8')
#Begin the segment
seg.begin()
#Because the reference to the segment is different than the
one at the Ignition gateway after begin was executed,
refresh it.
seg = system.mes.getActiveSegment('[global]\Dressings
Inc\California\Assembly\PS Assembly', 'Load Assembly Tray')
#Create new sublot
#Add custom properties
#Custom property definition format: {custom_property_name:
[ignition_data_type, value, description, units,
production_visible, required}
cp = {'Width' : ['Int4', 1020, 'Width of housing', 'mm', Tru
e, True], 'Height' : ['Int4', 800, 'Height of housing', 'mm'
, True, True]}
seg.addSublot('Housing', 'SN 1234', cp)
#Create second sublot with different custom property values
cp = {'Width' : ['Int4', 1025, 'Width of housing', 'mm', Tru
e, True], 'Height' : ['Int4', 790, 'Height of housing', 'mm'
, True, True]}
seg.addSublot('Housing', 'SN 2345', cp)
seg.update()
```

addSublots(materialPropertyName, countToAdd)

Description

These script functions are used to add multiple sublots to active segments and the different versions provide various methods to do so. Sublots are represented by MESMaterialSublot objects which corresponds to the ISA-95 Material Sublot objects. MESMaterialSublot objects must be children of a MESMaterialLot object, event though the lot information may not be needed. Usually, when production details are maintained for serialized items moving through production as groups, sublots are used. In cases where each item moves independently through production, then just material lots can be used for each serialized item.

Notice

In order for material sublots to be added, the Enable Sublots setting of the segment material property must be set to true.

Description

This version of the addSublots script function is used to create a specified quantity of new material sublots. For each sublot object created, the CreateSerialNumber event of the MaterialSublot object will be executed and a serial number will automatically be assigned. This serial number, which is the name of the MaterialSublot object, can be changed prior updating the segment.

Syntax

addSublots(materialPropertyName, countToAdd)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

Integer countToAdd - The number of sub lots to add.

- Returns

List<MESMaterialSublot> - A list of new MESMaterialSublot objects. The size of the list will match the countToAdd parameter.

- Scope

All

Code Examples

Code Snippet

```
#Create a new segment
seg = system.mes.createSegment('Load Assembly Tray', '[globa
l]\Dressings Inc\California\Assembly\PS Assembly', False)
```

```
#Assign the material resources
seg.setMaterial('Housing', 'Housing', 'Assembly Tray 8')
#Begin the segment
seg.begin()
#Because the reference to the segment is different than the
one at the Ignition gateway after begin was executed,
refresh it.
seg = system.mes.getActiveSegment('[global]\Dressings
Inc\California\Assembly\PS Assembly', 'Load Assembly Tray')
#Create new sublots
sublotList = seg.addSublots('Housing', 5)
for index in range(sublotList.size()):
    #Print the automatically generated serial number
    print sublotList.get(index).getName()
seg.update()
```

getSubLot

Description

Get an existing sublot that is associated with a segment. Sublots are represented by MESMaterialSublot objects and correspond to the ISA-95 Material Sublot objects.

Syntax

getSublot(materialPropertyName, sublotName)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

String sublotName - The name of an existing sublot to return.

- Returns

MESMaterialSublot - The MESMaterialSublot object.

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Get the current operation being run at the equipment
oper = system.mes.getCurrentOperation(eqPath)
#Get the active segment running under the operation
segName = oper.getActiveSegmentName()
seg = oper.getActiveSegment(segName)
#Get the sublot with serial number SN1823
sublot = seg.getSublot('Housing', 'SN1823')
#Do something with the sublot
sublot.setPropertyValue('Width', '1002')
#Don't forget to save the changes to the sublot
system.mes.saveMESObject(sublot)
```

Base Object

The Material Sublot is derived from the MESAbstractObject and inherits all the exposed properties, methods and events for that object.

Script Functions

The following script functions exist for the Material Sublot.

getLocationLink

Description

Get the link to the location of the material sublot.

Syntax

getLocationLink()

- Parameters

None

- Returns

The MES Object Link representing the location.

- Scope

All

getMaterialDefLink

Description

Get the link to the definition of the material sublot.

Syntax

getMaterialDefLink()

- Parameters

None

- Returns

The MES Object Link representing the definition.

- Scope

All

getName

Description

Gets the name of corresponding material sublot.

Syntax

getName()

- Parameters

None

- Returns

String name - The name of the corresponding material sublot.

- Scope

All

Methods

Beside the common MESAbstractObject methods, the following methods exist for the Material Sublots.

getLocationLink

Description

Get the link to the location of the material sublot.

Syntax

getLocationLink()

- Parameters

None

- Returns

The MES Object Link representing the location.

- Scope

All

getMaterialDefLink

Description

Get the link to the definition of the material sublot.

Syntax

getMaterialDefLink()

- Parameters

None

- Returns

The MES Object Link representing the definition.

- Scope

All

getName

Description

Gets the name of corresponding material sublot.

Syntax

getName()

- Parameters

None

- Returns

String name - The name of the corresponding material sublot.

- Scope

All

Events

Besides the common MES object events, the following events exist for Material Sublot objects.

Setting Name	Description
New	

Setting Name	Description
	The event is fired when a new instance of a Material Sub-Lot object is created. If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()
CreateSerialNumber	This event is run every time a Material Sublot object is requested to create a new serial number. This event provides a method to intercept the generation of serial numbers so that the format of the serial number or even the number itself can be modified. For systems that retrieve serial numbers from a ERP or other system, this event will allow obtaining a serial number from it. Script in this event can also read from a block of available serial numbers that are maintained in a database.

Properties

Property values can be accessed and changed for an object by using the getPropertyValue() and setPropertyValue() method.

Example

```
eqPath = '[global]\Dressings Inc\California\Raw Materials\Unload
Station 1'

#Get the current operation being run at the equipment
oper = system.mes.getCurrentOperation(eqPath)

#Get the active segment running under the operation
segName = oper.getActiveSegmentName()
seg = oper.getActiveSegment(segName)

#Get the sublot with serial number SN1823
sublot = seg.getSublot('Housing', 'SN1823')

#Do something with the sublot
sublot.setPropertyValue('Width', '1002')

#Don't forget to save the changes to the sublot
system.mes.saveMESObject(sublot)
```

Besides the common core properties, the following properties exist for Material Sublots.

Setting Name	R/W	Description
Lot Assembly Type	Read	Currently this is here for reference and is included to align with the ISA-95 standard and will become significant in the next phase of the Track and Trace Module.
Sublot Status	Read /Write	The status of the Material Sublot object. When a Material Sublot object is currently being processed by a Response Segment, this will be set to Active. When the Response Segment is ended, this will be set to the Final Lot Status setting in the Operation Segment configuration. If a Final Lot Status is not defined, it will be set to the default status of Completed.

MESLotAvailabilityStatusTypes

This object is used for filtering the results based on the lot status. The available lot status types are:

- Available
- Used
- Both

Lot status can also be set with setPropertyValue('LotAvailabilityStatus', 'Both') script function

```
#Load MES object
obj = system.mes.loadMESObject('Box', 'MaterialDef')

#Get and print the current name of the MES object
#Notice either method can be used to read the Name property
print obj.getName()
print obj.getPropertyValue('Name')
#Change the name of the MES object
obj.setPropertyValue('Name', 'Empty Box')

#Set the lot status
obj.setPropertyValue('LotAvailabilityStatus', 'Available')

#Don't forget to save the MES object after changing property
values
system.mes.saveMESObject(obj)
```

MES Analysis Results

Base Object

The MES Analysis Results object is derived from the MESAbstractObject and inherits all the exposed properties, methods and events for that object.

Scripting Functions

The following scripting functions return a MES Analysis Results object ...

system.mes.executeAnalysis(beginDate, endDate, settings) system.mes.executeAnalysis (beginDate, endDate, savedSettingsName)

Methods

Beside the common MESAbstractObject methods, the following methods exist for the MES Analysis Results object .

addMessage(message)

Description

Adds a message to the analysis results.

Syntax

addMessage(message)

- Parameters

String message - The message to be added.

- Returns

Nothing

- Scope

All

getDrillDownOptions()

Description

Gets the drill down options for the analysis results.

Syntax

getDrillDownOptions()

- Parameters

None

- Returns

List<AbstractValueItemInfo> options - The drill down options for the analysis results.

- Scope

All

getExecutionDurationMS()

Description

Gets the duration of execution in milliseconds.

Syntax

getExecutionDurationMS()

- Parameters

None

- Returns

Long executionDurationMS - The duration of execution in milliseconds.

- Scope

All

setDrillDownOptions(List<AbstractValueItemInfo> drillDownOptionMap)

Description

Sets a list of drill down options for this analysis results.

Syntax

setDrillDownOptions(List<AbstractValueItemInfo> drillDownOptionMap)

- Parameters

List<AbstractValueItemInfo> options - Drill down options for the analysis results.

- Returns

Nothing

- Scope

All

MES Analysis Settings

Base Object

The MES Analysis Settings object is derived from the MESAbstractObject and inherits all the exposed properties, methods and events for that object.

Scripting Functions

The following scripting functions return a MES Analysis Settings object ...

- system.mes.createMESAnalysisSettings
- system.mes.getMESAnalysisSettings
- system.mes.getMESAnalysisSettingsList

Methods

Beside the common MESAbstractObject methods, the following methods exist for the MES Analysis Settings object .

addParameter(parameterName)

Description

Adds a parameter to the analysis settings.

Syntax

addParameter(parameterName)

- Parameters

String parameterName - Name of the parameter to be added.

- Returns

Nothing

- Scope

All

addSecurityRole(roleName)

Description

Adds a security role to the analysis settings.

Syntax

addSecurityRole(roleName)

- Parameters

String roleName - The name of security role to be added.

- Returns

Nothing

- Scope

All

canUserExecute(user)

Description

Checks whether an MES user is allowed to execute the analysis settings.

Syntax

canUserExecute(user)

- Parameters

MESUser user - The MES user for which the canUserExecute property is checked for.

- Returns

boolean True if the MES user can execute analysis settings and False otherwise.

- Scope

All

canUserModify(user)

Description

Checks whether an MES user is allowed to modify the analysis settings.

Syntax

canUserModify(user)

- Parameters

MESUser user - The MES user for which the canUserModify property is checked for.

- Returns

boolean True if the MES user can modify analysis settings and False otherwise.

- Scope

All

getDataPointList()

Description

Gets the list of data points associated with the analysis settings.

Syntax

getDataPointList()

- Parameters

None

- Returns

List<String> dataPointList - A list of data points for the analysis settings.

- Scope

All

getDataPoints()

Description

Gets the data points associated with the analysis settings.

Syntax

getDataPoints()

- Parameters

None

- Returns

String dataPoints - Data points for the analysis settings.

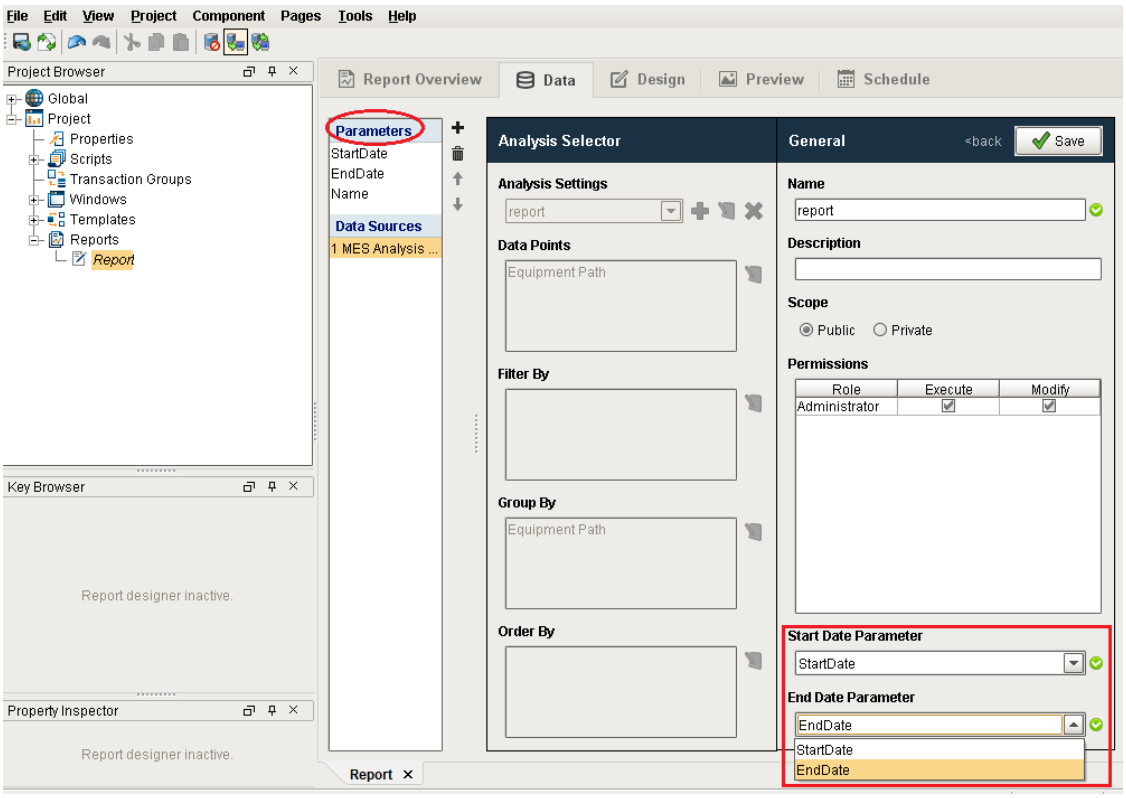
- Scope

All

getEndDateParameterName()

Info

Analysis data can be filtered using Ignition report parameters. They can be selected from the drop down menu of the analysis sector (only for reports) as shown below.



Description

Gets the end date parameter name for the MES analysis.

Syntax

getEndDateParameterName()

- Parameters

None

- Returns

String parameterName - The end date parameter name for the MES analysis (reports).

- Scope

All

getFilterExpression()

Description

Gets the filter expression for the analysis settings.

Syntax

getFilterExpression()

- Parameters

None

- Returns

String expression - The filter expression for the analysis settings.

- Scope

All

getGroupBy()

Description

Gets the GroupBy for this analysis settings.

Syntax

getGroupBy()

- Parameters

None

- Returns

String groupBy - GroupBys defined in the analysis settings.

- Scope

All

getGroupByList()

Description

Gets the list of GroupBys defined in the analysis settings.

Syntax

getGroupByList()

- Parameters

None

- Returns

List<String> groupByList - The list of GroupBys defined in the analysis settings.

- Scope

All

getIncludeDrillDownOptions()

Description

Gets the includeDrillDownOptions property value for the analysis settings.

Syntax

getIncludeDrillDownOptions()

- Parameters

None

- Returns

boolean includeOptions - True if the drill down options are included and False otherwise.

- Scope

All

getOrderBy()

Description

Gets the OrderBys defined in the analysis settings.

Syntax

getOrderBy()

- Parameters

None

- Returns

String orderBy - The OrderBys defined in the analysis data.

- Scope

All

getOrderByList()

Description

Gets the list of OrderBys defined in the analysis settings.

Syntax

getOrderByList()

- Parameters

None

- Returns

List<String> orderByList - The list of OrderBys defined in the analysis settings.

- Scope

All

getParameterCount()

Description

Gets the number of parameter for the analysis settings.

Syntax

getParameterCount()

- Parameters

None

- Returns

int count - The parameter count for the analysis settings.

- Scope

All

getParameter(index)

Description

Gets the analysis parameter corresponding to the given index.

Syntax

getParameter(index)

- Parameters

int index - The index representing the parameter.

- Returns

AnalysisParameterProperty analysisParameter - The analysis parameter complex property defined by the specified index.

- Scope

All

getParameter(parameterName)

Description

Gets the analysis parameter.

Syntax

getParameter(parameterName)

- Parameters

String parameterName - The name of the parameter to be returned.

- Returns

AnalysisParameterProperty analysisParameter - The analysis parameter complex property.

- Scope

All

getSecurityRoleCount()

Description

Gets the security role count for the analysis settings.

Syntax

getSecurityRoleCount()

- Parameters

None

- Returns

int count - The count of security roles associated with this analysis settings.

- Scope

All

getSecurityRole(index)

Description

Gets the security role corresponding to the index parameter.

Syntax

getSecurityRole(index)

- Parameters

int index - The security role index.

- Returns

AnalysisSecurityProperty analysisSecurity - The analysis security complex property with the security role details.

- Scope

All

getSecurityRole(roleName)

Description

Gets the security role corresponding to the role name parameter.

Syntax

getSecurityRole(roleName)

- Parameters

String roleName - The role name to return the security role for.

- Returns

AnalysisSecurityProperty analysisSecurity - The analysis security complex property with the security role details.

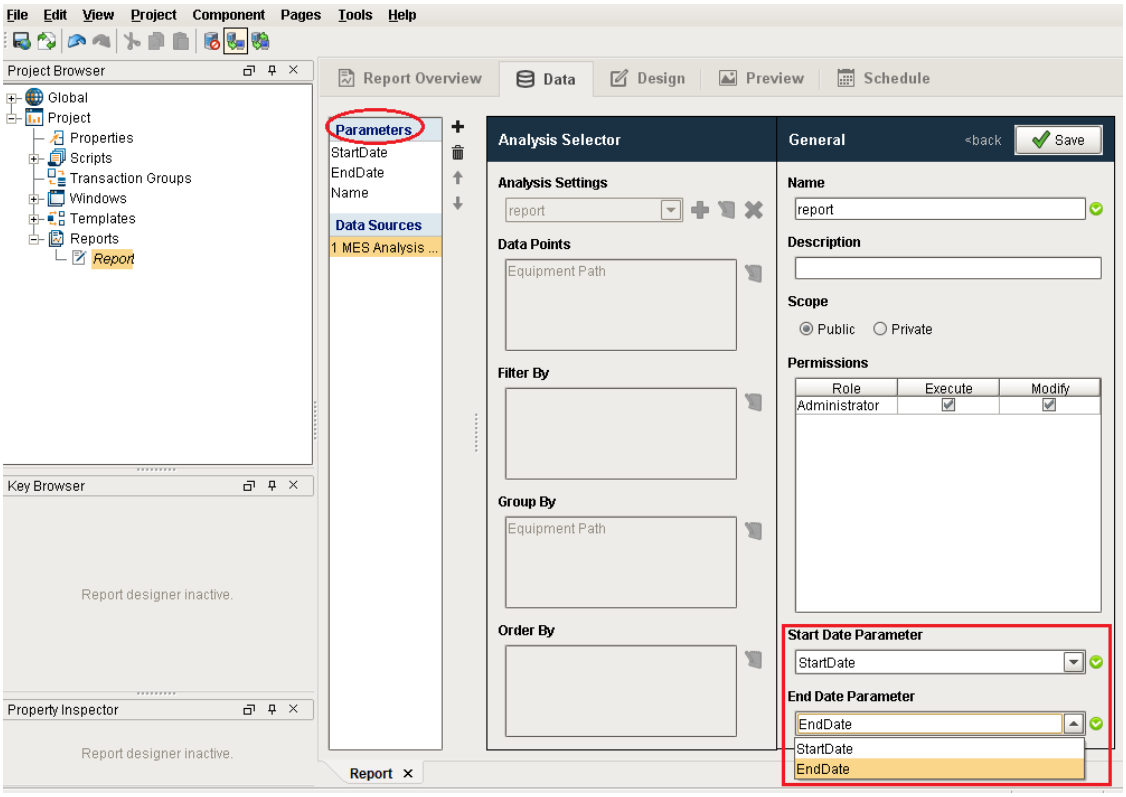
- Scope

All

getStartDateParameterName()

Info

Analysis data can be filtered using Ignition report parameters. They can be selected from the drop down menu of the analysis sector (only for reports) as shown below.



Description

Gets the start date parameter name from the MES analysis (report).

Syntax

getStartDateParameterName()

- Parameters

None

- Returns

String parameterName - The start date parameter defined in the MES analysis.

- Scope

All

hasDataPoints()

Description

Checks whether there exist any data points for the analysis settings.

Syntax

hasDataPoints()

- Parameters

None

- Returns

boolean - True if there exist any data points for the analysis settings and False otherwise.

- Scope

All

removeParameter(parameterName)

Description

Removes the specific parameter from the analysis settings.

Syntax

removeParameter(parameterName)

- Parameters

String parameterName - Name of the parameter to be removed.

- Returns

Nothing

- Scope

All

removeSecurityRole(roleName)

Description

Removes a security role from the analysis settings.

Syntax

removeSecurityRole(roleName)

- Parameters

String roleName - The name of security role that is to be removed.

- Returns

Nothing

- Scope

All

setDataPoints(dataPoints)

Description

Sets the data points for this analysis settings.

Syntax

setDataPoints(dataPoints)

- Parameters

String dataPoints - The data points to set for the analysis settings.

- Returns

Nothing

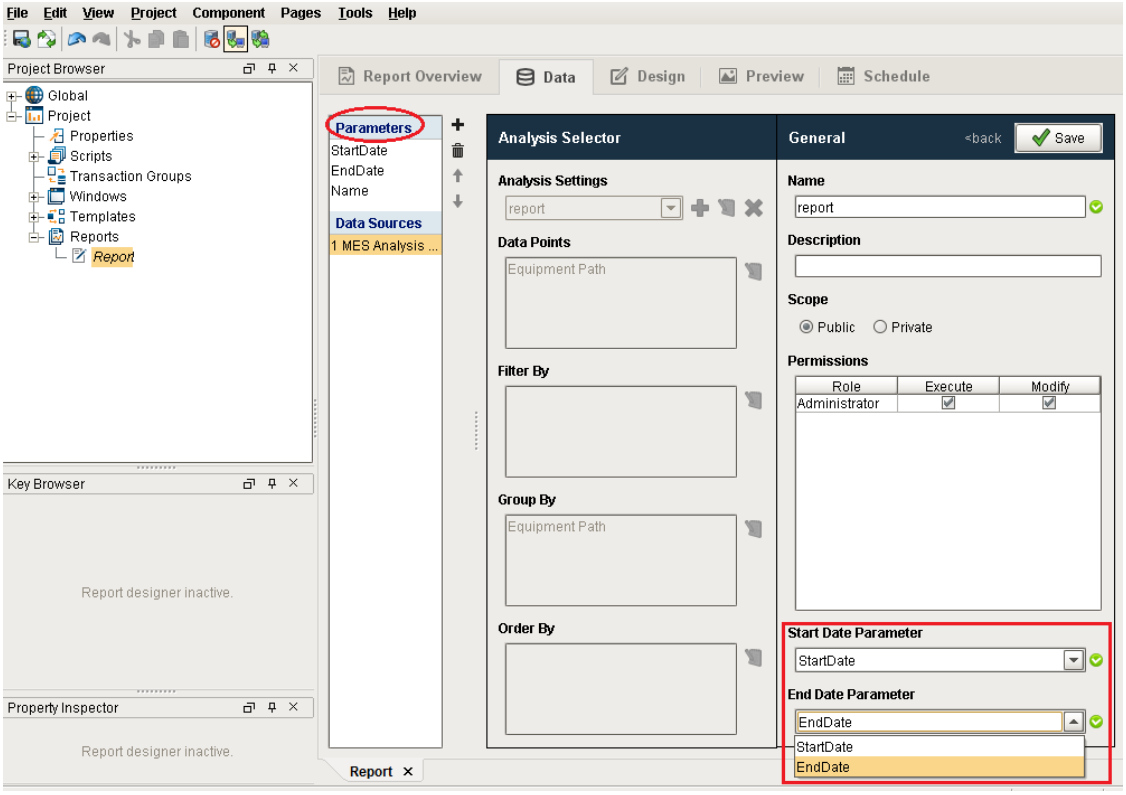
- Scope

All

setEndDateParameterName(parameterName)

Info

Analysis data can be filtered using Ignition report parameters. They can be selected from the drop down menu of the analysis sector (only for reports) as shown below.



Description

Sets the end date parameter name for the MES analysis settings (reports).

Syntax

setEndDateParameterName(parameterName)

- Parameters

String parameterName - The end date parameter name to set for the analysis settings.

- Returns

Nothing

- Scope

All

setFilterExpression(expression)

Description

Sets the filter expression for the analysis settings.

Syntax

setFilterExpression(expression)

- Parameters

String expression - The filter expression to be set for.

- Returns

Nothing

- Scope

All

setGroupBy(groupBy)

Description

Sets the GroupBy to group the analysis data.

Syntax

setGroupBy(groupBy)

- Parameters

String groupBy - The factors to group the analysis data.

- Returns

Nothing

- Scope

All

setGroupBy(groupBy)

Description

Sets the factors (GroupBy) to group the analysis data.

Syntax

setGroupBy(groupBy)

- Parameters

List<String> groupBy - The list of GroupBys separated by commas.

- Returns

Nothing

- Scope

All

setIncludeDrillDownOptions(includeOptions)

Description

Sets the includeDrillDownOptions property value for the analysis settings.

Syntax

setIncludeDrillDownOptions(includeOptions)

- Parameters

boolean includeOptions - Set to True if you like to include the drill down options and False otherwise.

- Returns

Nothing

- Scope

All

setOrderBy(orderBy)

Description

Sets the OrderBy for the analysis data.

Syntax

setOrderBy(orderBy)

- Parameters

String orderBy - The orderBy to set for the analysis data.

- Returns

Nothing

- Scope

All

setOrderBy(orderBy)

Description

Sets a list of OrderBys for the analysis data.

Syntax

setOrderBy(orderBy)

- Parameters

List<String> orderBy - The list of OrderBys for the analysis data.

- Returns

Nothing

- Scope

All

setParameterDataType(parameterName, parameterDataType)

Description

Sets the datatype for the analysis parameter.

Syntax

setParameterDataType(parameterName, parameterDataType)

- Parameters

String parameterName - Name of the parameter to set the data type for.

String parameterDataType - Data type to set for the parameter.

- Returns

Nothing

- Scope

All

setSecurityRoleRights(roleName, canExecute, canModify)

Description

Sets the rights to security role.

Syntax

setSecurityRoleRights(roleName, canExecute, canModify)

- Parameters

String roleName - The name of the security role to set the security rights for.

boolean canExecute - Set to True if this security role can execute the analysis and False otherwise.

boolean canModify - Set to True if this security role can modify the analysis and False otherwise.

- Returns

Nothing

- Scope

All

MES Inventory Filter

Object Description

A MESInventoryFilter object used to query inventory information and is required by certain script methods. Because inventory results may be specific to known materials, locations, etc., this object is used to hold the filter settings.

Scripting Functions

This script function can be used to create an **MESInventoryFilter** object.

system.mes.inventory.filter.createFilter()

Description

Returns a new instance of a MESInventoryFilter object for that properties can be set on. This is typically used when a script function requires a MESInventoryFilter object as a parameter.

Syntax

system.mes.inventory.filter.createFilter()

- Parameters

None

- Returns

MESInventoryFilter - A new instance of a MESInventoryFilter object.

Example

Code Snippet

Code Snippet

```
from java.util import Calendar

filter = system.mes.inventory.filter.createFilter()
filter.setIncludeCompleteLots(True)
filter.setEquipmentClassName('Vinegar Tanks')
beginCal = Calendar.getInstance()
beginCal.add(Calendar.DAY_OF_MONTH, -30)
filter.setBeginDateTime(beginCal)
endCal = Calendar.getInstance()
filter.setEndDateTime(endCal)
results = system.mes.getInventory(filter)
```

```
for rowIndex in range(results.getRowCount()):
    print str(results.getValueAt(rowIndex, 0))
```

Methods

Beside the common MESAbstractObject methods, the following methods exist for the MES Inventory Filter Object.

getBeginDateTime()

Description

Get the beginning date and time to limit the results to return.

Syntax

getBeginDateTime()

- Parameters

None

- Returns

Calendar The beginning date and time to filter results.

getCustomLotStatus()

Description

Get the custom lot status of results to return.

Syntax

getCustomLotStatus()

- Parameters

None

- Returns

String The custom lot status value.

getEndDateTime()

Description

Get the ending date and time to limit the results to return.

Syntax

getEndDateTime()

- Parameters

None

- Returns

Calendar The ending date and time to filter results.

getEquipmentClassName()

Description

Get the lot equipment class name used to filter the results.

Syntax

getEquipmentClassName()

- Parameters

None

- Returns

String The lot equipment class name.

Code Examples

Code Snippet

```
#Example
filter = system.mes.inventory.filter.createFilter()
filter.setIncludeActiveLots(True)
filter.setEquipmentClassName('Vinegar Tanks')
print filter.getEquipmentClassName()
```

Output

```
Vinegar Tanks
```

getEquipmentClassUUID()

Description

Get the lot equipment class UUID used to filter the results.

Syntax

getEquipmentClassUUID()

- Parameters

None

- Returns

String The lot equipment class UUID.

getEquipmentPath()

Description

Get the lot equipment path used to filter the results.

Syntax

getEquipmentPath()

- Parameters

None

- Returns

String The lot equipment path.

getEquipmentUUID()

Description

Get the lot equipment UUID used to filter the results.

Syntax

getEquipmentUUID()

- Parameters

None

- Returns

String The lot equipment UUID.

getMaterialClassName()

Description

Get the material class name used to filter the results.

Syntax

getMaterialClassName()

- Parameters

None

- Returns

String The material class name.

getMaterialClassUUID()

Description

Get the material class UUID used to filter the results.

Syntax

getMaterialClassUUID()

- Parameters

None

- Returns

String The material class UUID.

getMaterialDefUUID()

Description

Get the material definition UUID used to filter the results.

Syntax

getMaterialDefUUID()

- Parameters

None

- Returns

String The material name UUID.

getMaterialNameFilter()

Description

Get the material name filter used to filter the results.

Syntax

getMaterialNameFilter()

- Parameters

None

- Returns

String The material name filter.

getPersonFirstName()

Description

Get the person first name used to filter the results.

Syntax

getPersonFirstName()

- Parameters

None

- Returns

String The person first name filter.

getPersonLastName()

Description

Get the person last name used to filter the results.

Syntax

getPersonLastName()

- Parameters

None

- Returns

String The person last name filter.

getPersonnelClassName()

Description

Get the personnel class name used to filter the results.

Syntax

getPersonnelClassName()

- Parameters

None

- Returns

String The personnel class name.

getPersonnelClassUUID()

Description

Get the personnel class UUID used to filter the results.

Syntax

getPersonnelClassUUID()

- Parameters

None

- Returns

String The personnel class UUID.

getPersonUUID()

Description

Get the person UUID used to filter the results.

Syntax

getPersonUUID()

- Parameters

None

- Returns

String The person UUID filter.

includeActiveLots()

Description

If True, lots or sublots currently being processed will be included in the results.

Syntax

includeActiveLots()

- Parameters

None

- Returns

Boolean If True, active lots will be return in the results.

Code Examples

Code Snippet

```
#Example
filter = system.mes.inventory.filter.createFilter()
filter.includeActiveLots()
```

includeCompleteLots()

Description

If True, lots that are complete will be included in the results.

Syntax

includeCompleteLots()

- Parameters

None

- Returns

Boolean If True, completed lots will be returned in the results.

Code Examples

Code Snippet

```
#Example
filter = system.mes.inventory.filter.createFilter()
filter.includeCompleteLots()
```

setBeginDateTime(beginDateTime)

Description

Set the beginning date and time to limit results to return. This applies to lots or sublots that are completed or have a custom lot status.

Syntax

setBeginDateTime(beginDateTime)

- Parameters

Calendar beginDateTime - Beginning date and time to filter results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
from java.util import Calendar
cal = Calendar.getInstance()
cal.add(Calendar.MONTH, -1)
beginDate = cal.getTime()
filter = system.mes.inventory.filter.createFilter()
filter.setBeginDateTime(beginDate)
```

setCustomLotStatus(customLotStatus)

Description

Set the custom lot status of results to return. If the Final Lot Status property in a resource definition of a Process Segment or Operations Segment is set to a custom lot status, it can be filtered with this property.

Syntax

setCustomLotStatus(customLotStatus)

- Parameters

String customLotStatus - Custom lot status value to filter results.

- Returns

Nothing

Code Examples

```
Code Snippet

#Example
filter = system.mes.inventory.filter.createFilter()
filter.setCustomLotStatus('Good')
```

setEndDateTime(endDateTime)

Description

Set the ending date and time to limit results to return. This applies to lots or sublots that are completed or have a custom lot status.

Syntax

setEndDateTime(endDateTime)

- Parameters

Calendar endDateTime - Ending date and time to filter results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
from java.util import Calendar
cal = Calendar.getInstance()
cal.add(Calendar.MONTH, -1)
endDate = cal.getTime()
filter = system.mes.inventory.filter.createFilter()
filter.setEndDateTime(endDate)
```

setEquipmentClassName(equipmentClassName)

Description

The results can be limited to only include lots or sublots that are or were stored at the equipment that are included in a material class that match this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Only one of the Equipment Class Name, Equipment Class UUID, Equipment Path or Equipment UUID properties can be specified at a time.

Example
Vinegar Storage Tanks.

Syntax

setEquipmentClassName(equipmentClassName)

- Parameters

String equipmentClassName - The lot equipment class name used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.inventory.filter.createFilter()
filter.setEquipmentClassName('Vinegar Tank?')
```

setEquipmentClassUUID(equipmentClassUUID)

Description

The results can be limited to only include lots or sublots that are or were stored at the equipment that are included in a material class that match this property. See UUIDs for more information.

Only one of the Equipment Class Name, Equipment Class UUID, Equipment Path or Equipment UUID properties can be specified at a time.

Syntax

setEquipmentClassUUID(equipmentClassUUID)

- Parameters

String equipmentClassUUID - The lot equipment class UUID used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.inventory.filter.createFilter()
filter.setIncludeActiveLots(True)
filter.setEquipmentClassUUID('a0a7991c-ee75-47d7-8c91-
b0e20e736ea9')
results = system.mes.getInventory(filter)
print filter.getEquipmentClassUUID()
```

Output

```
a0a7991c-ee75-47d7-8c91-b0e20e736ea9
```

setEquipmentPath(equipmentPath)

Description

The results can be limited to only include lots or sublots that are or were stored at the equipment that match this property. See Equipment for more information on equipment paths.

Only one of the Equipment Class Name, Equipment Class UUID, Equipment Path or Equipment UUID properties can be specified at a time.

Syntax

setEquipmentPath(equipmentPath)

- Parameters

String equipmentPath - The lot equipment path used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.inventory.filter.createFilter()
filter.setEquipmentPath('[global]\My
Enterprise\California\Storage\Vinegar Tanks\Vinegar Tank 1')
```

setEquipmentUUID(equipmentUUID)

Description

The results can be limited to only include lots or sublots that are or were stored at the equipment that match this property. See UUIDs for more information.

Syntax

setEquipmentUUID(equipmentUUID)

- Parameters

String equipmentClassUUID - The lot equipment UUID used to filter the results.

- Returns

Nothing

Code Examples

```
Code Snippet

#Example
filter = system.mes.inventory.filter.createFilter()
filter.setEquipmentUUID('8da06ff8-2922-4e0c-a01a-e7cda6899a0e')
```

setIncludeActiveLots(includeActiveLots)

Description

If set to True, lots or sublots that are actively being processed will be included in the results.

Syntax

setIncludeActiveLots(includeActiveLots)

- Parameters

Boolean includeActiveLots - If True, include active lots or sublots in results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.inventory.filter.createFilter()
filter.setIncludeActiveLots(True)
```

setIncludeCompleteLots(includeCompleteLots)

Description

If set to True, lots or sublots that are completed will be included in the results.

Syntax

setIncludeCompleteLots(includeCompleteLots)

- Parameters

Boolean includeActiveLots - If True, include completed lots or sublots in results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.inventory.filter.createFilter()
```

```
filter.setIncludeCompleteLots(True)
```

setMaterialClassName(materialClassName)

Description

The results can be limited to only include lots or sublots that the associated material is included in a material class that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Only one of the Material Class Name, Material Class UUID, Material Definition Name, Material Definition UUID properties can be specified at a time.

Example:
Vinegar

Syntax

setMaterialClassName(materialClassName)

- Parameters

String materialClassName - The material class name used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.inventory.filter.createFilter()
filter.setMaterialClassUUID('a3809970-2c99-4fce-a3dc-
14f8e80155a5')
```

setMaterialClassUUID(materialClassUUID)

Description

The results can be limited to only include lots or sublots that the associated material is included in a material class that matches this property. See UUIDs for more information.

Only one of the Material Class Name, Material Class UUID, Material Definition Name, Material Definition UUID properties can be specified at a time.

Syntax

setMaterialClassUUID(materialClassUUID)

- Parameters

String materialClassUUID - The material class name used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.inventory.filter.createFilter()
filter.setMaterialClassUUID('5acf3c9f-2789-44af-888f-
fce08d9972a7')
```

setMaterialDefName(materialDefName)

Description

The results can be limited to only include lots or sublots that the associated material matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Only one of the Material Class Name, Material Class UUID, Material Definition Name, Material Definition UUID properties can be specified at a time.

Example
Balsamic

Syntax

setMaterialDefName(materialDefName)

- Parameters

String materialDefName - The material definition name filter used to filter the results.

- Returns

Nothing

Code Examples

```
Code Snippet

#Example
filter = system.mes.inventory.filter.createFilter()
filter.setMaterialDefName('Raw Balsamic Vinegar')
```

setMaterialDefUUID(materialDefUUID)

Description

The results can be limited to only include lots or sublots that the associated material matches this property. See UUIDs for more information.

Only one of the Material Class Name, Material Class UUID, Material Definition Name, Material Definition UUID properties can be specified at a time.

Syntax

setMaterialDefUUID(materialDefUUID)

- Parameters

String materialDefUUID - The material definition UUID used to filter the results.

- Returns

Nothing

Code Examples

```
Code Snippet

#Example
filter = system.mes.inventory.filter.createFilter()
filter.setMaterialDefUUID('8ea5bd6b-80ec-484f-98b4-
2de7d6d0724a')
```

setPersonnelClassName(personnelClassName)

Description

The results can be limited to only include lots or sublots that are or were handled by personnel that are included in a personnel class that match this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Only one of the Personnel Class Name, Personnel Class UUID or Person First Name and Person Last Name combination properties can be specified at a time.

Example
Unload Operator

Syntax

setPersonnelClassName(personnelClassName)

- Parameters

String personnelClassName - The personnel class name used to filter the results.

- Returns

Nothing

Code Examples

```
Code Snippet

#Example
filter = system.mes.inventory.filter.createFilter()
filter.setPersonnelClassName('Operator')
```

setPersonnelClassUUID(personnelClassUUID)

Description

The results can be limited to only include lots or sublots that are or were handled by personnel that are included in a personnel class that match this property. See UUIDs for more information.

Only one of the Personnel Class Name, Personnel Class UUID or Person First Name and Person Last Name combination properties can be specified at a time.

Syntax

setPersonnelClassUUID(personnelClassUUID)

- Parameters

String personnelClassUUID - The personnel class UUID used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.inventory.filter.createFilter()
filter.setPersonnelClassUUID('865d2c95-f06c-4cce-9625-
3631e465e904')
```

setPersonFirstName(personFirstName)

Description

The results can be limited to only include lots or sublots that are or were handled by personnel that match this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setPersonFirstName(personFirstName)

- Parameters

String personFirstName - The person first name used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.inventory.filter.createFilter()
filter.setPersonFirstName('Mark')
```

setPersonLastName(personLastName)

Description

The results can be limited to only include lots or sublots that are or were handled by personnel that match this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setPersonLastName(personLastName)

- Parameters

String personLastName - The person last name used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.inventory.filter.createFilter()
filter.setPersonLastName('West')
```

setPersonUUID(personUUID)

Description

The results can be limited to only include lots or sublots that are or were handled by personnel that match this property. See UUIDs for more information.

Syntax

setPersonUUID(personUUID)

- Parameters

String personUUID - The person UUID used to filter the results.

- Returns

Nothing

Code Examples

```
Code Snippet

#Example
filter = system.mes.inventory.filter.createFilter()
filter.setPersonUUID('654e3245-2f95-4d10-8fda-53e22400482e')
```

MES Lot Filter

Object Description

The MESLotFilter object is used to help when searching for lots or sublots and is required for certain script methods. Lot or sublot search results can be limited by using the MESLotFilter properties to narrow down the MES lots to return when using the system.mes.getLotList script function.

The MESLotFilter object doesn't have a option to set the lot sequence number. The loadMaterialLotLink and loadMaterialLot script function will return the last lot if the lot sequence parameter is less than 0. All function using the MESLotFilter object return a list of lots and will include all lots that match the settings in the MESLotFilter object. So not just the last lot, but all that apply.

Scripting Functions

The following function can be used to create MESLotFilter.

system.mes.lot.filter.createFilter()

Description

Returns a new instance of a MESLotFilter object for that properties can be set on. This is typically used when a script function requires a MESLotFilter object as a parameter.

Syntax

system.mes.lot.filter.createFilter()

- Parameters

None

- Returns

A new instance of a MESLotFilter object.

Code Snippet

```
from java.util import Calendar
filter = system.mes.lot.filter.createFilter()
filter.setModeName('LOT')
filter.setIncludeInactiveLots(True)
beginCal = Calendar.getInstance()
beginCal.add(Calendar.DAY_OF_MONTH, -30)
filter.setBeginDateTime(beginCal)
endCal = Calendar.getInstance()
filter.setEndDateTime(endCal)
results = system.mes.getLotList(filter)
for link in results:
    print link.getName()
```

Example

If the MESLotFilter was not provided, then each option would have to be passed as a parameter in the system.mes.getLotList method. There are over a dozen options to filter lot on. This would make it very difficult to use because the line of script would look something like the following:

Code Example 1

```
#Cumbersome method that is NOT used:
system.mes.getLotList('', '000*', '', '', '', '', '', '', '', '',
'', beginDate, endDate,'', '', '')
```

Code Example 2

```
#Instead, using the MESLotFilter object the script look like:
filter = system.mes.lot.filter.createFilter()
filter.setLotNameFilter('000*')
filter.setBeginDateTime(beginDate)
filter.setEndDateTime(endDate)
list = system.mes.getLotList(filter)
```

The second example is the supported method and is much more readable.

Methods

The following methods exist for the MES Lot Filter Object.

getBeginDateTime()

Description

Get the beginning date and time to limit the results to return.

Syntax

getBeginDateTime()

- Parameters

None

- Returns

Calendar The beginning date and time to filter results.

Info

Custom Property Value Filter

Description

The results can be limited to only include items that have a custom property expressions defined by this property that evaluates to true.Example Kind > 3.

getCustomPropertyValueFilter()

Description

Get the list of MESPropertyValueFilter used to filter the results.

Syntax

getCustomPropertyValueFilter()

- Parameters

None

- Returns

List of MES Property Value Filter - The custom property value filter containing information about MESObjectTypes, propertyPath, etc .

getEndDateTime()

Description

Get the ending date and time to limit the results to return.

Syntax

getEndDateTime()

- Parameters

None

- Returns

Calendar The ending date and time to filter results.

getLotEquipmentClassFilter()

Description

Get the lot equipment class filter used to filter the results.

Syntax

getLotEquipmentClassFilter()

- Parameters

None

- Returns

String The lot equipment class filter.

getLotEquipmentNameFilter()

Description

Get the lot equipment name filter used to filter the results.

Syntax

getLotEquipmentNameFilter()

- Parameters

None

- Returns

String The lot equipment name filter.

getLotNameFilter()

Description

Get the lot name filter used to filter the results.

Syntax

getLotNameFilter()

- Parameters

None

- Returns

String The lot name filter.

getLotStatusFilter()

Description

Get the custom lot status of results to return.

Syntax

getLotStatusFilter()

- Parameters

None

- Returns

String - The custom lot status value.

getMaterialClassFilter()

Description

Get the material class filter used to filter the results.

Syntax

getMaterialClassFilter()

- Parameters

None

- Returns

String The material class filter.

getMaterialNameFilter()

Description

Get the material name filter used to filter the results.

Syntax

getMaterialNameFilter()

- Parameters

None

- Returns

String The material name filter.

getMaxResults()

Description

Get the maximum number of results to that will returned.

Syntax

getMaxResults()

- Parameters

None

- Returns

Integer The maximum number of items to return.

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
print filter.getMaxResults()
```

Output

```
100
```

getModeName()

Description

Get the type of results to return. It can be return results for lots (batches) of material or serialized items (sublots).

Syntax

getModeName()

- Parameters

None

- Returns

String Name - The name of the type of results to return.

Code Examples

Code Snippet

```
#Prints the mode names.
```

```
filter = system.mes.lot.filter.createFilter()
filter.setIncludeActiveLots(True)
print filter.getModeName()
```

Output

```
LOT
```

getOperationNameFilter()

Description

Get the operation name filter used to filter the results.

Syntax

getOperationNameFilter()

- Parameters

None

- Returns

String The operation name filter.

getPersonnelClassFilter()

Description

Get the personnel class filter used to filter the results.

Syntax

getPersonnelClassFilter()

- Parameters

None

- Returns

String The personnel class filter.

getPersonnelNameFilter()

Description

Get the personnel name filter used to filter the results.

Syntax

getPersonnelNameFilter()

- Parameters

None

- Returns

String The P ersonnelNameFilter.

getSegmentEquipmentClassFilter()

Description

Get the segment equipment class filter used to filter the results.

Syntax

getSegmentEquipmentClassFilter()

- Parameters

None

- Returns

String The segment equipment class filter.

getSegmentEquipmentNameFilter()

Description

Get the segment equipment name filter used to filter the results.

Syntax

getSegmentEquipmentNameFilter()

- Parameters

None

- Returns

String The lot equipment name filter.

getSegmentNameFilter()

Description

Get the segment name filter used to filter the results.

Syntax

getSegmentNameFilter()

- Parameters

None

- Returns

String The segment name filter.

getSublotNameFilter()

Description

Get the sublot name filter used to filter the results.

Syntax

getSublotNameFilter()

- Parameters

None

- Returns

String The sublot name filter.

hasCustomPropertyValuefilter()

Description

Checks to see if a custom property value filter exists for the given lot.

Syntax

hasCustomPropertyValuefilter()

- Parameters

None

- Returns

True, if there exist a custom property value filter .

- Scope

All

includeActiveLots()

Description

If True, lots or sublots currently being processed will be included in the results.

Syntax

includeActiveLots()

- Parameters

None

- Returns

Boolean If True, active lots will be return in the results.

Code Examples

```
Code Snippet

#Example
filter = system.mes.lot.filter.createFilter()
filter.includeInactiveLots()
```

includeInactiveLots()

Description

If True, lots or sublots that are complete will be included in the results.

Syntax

includeInactiveLots()

- Parameters

None

- Returns

Boolean - If True, completed lots will be return in the results.

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.includeInactiveLots()
```

setBeginDateTime(beginDateTime)

Description

Set the beginning date and time to limit results to return. This applies to lots or sublots that are completed or have a custom lot status.

Syntax

setBeginDateTime(beginDateTime)

- Parameters

Calendar beginDateTime - Beginning date and time to filter results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
from java.util import Calendar
filter = system.mes.lot.filter.createFilter()
begin = Calendar.getInstance()
begin.add(Calendar.MONTH, -1)
filter.setBeginDateTime(begin)
```

setCustomPropertyValueFilter(customPropertyValueFilter)

Description

Set the custom property filter expressions to filter the results. If a custom property of a MES object matches an expression in this list, then it will be included in the results. Use system.mes.object.filter.parseCustomPropertyValueFilter() script function to create the list of MES Property Value Filter objects.

Syntax

setCustomPropertyValueFilter(customPropertyValueFilter)

- Parameters

List of MES Property Value Filter customPropertyValueFilter - The custom property value list to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
filter = system.mes.object.filter.createFilter()
list = system.mes.object.filter.parseCustomPropertyValueFilter(
'pH > 5.0,Width = 2.5')
filter.setCustomPropertyValueFilter(list)
```

setEndDateTime(endDateTime)

Description

Set the ending date and time to limit results to return. This applies to lots or sublots that are completed or have a custom lot status.

Syntax

setEndDateTime(endDateTime)

- Parameters

Calendar endDateTime - Ending date and time to filter results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example using current time as endTime
from java.util import Calendar
filter = system.mes.lot.filter.createFilter()
endTime = Calendar.getInstance()
filter.setEndDateTime(endTime)
```

setIncludeActiveLots(includeActiveLots)

Description

If set to True, lots or sublots that are actively being processed will be included in the results.

Syntax

setIncludeActiveLots(includeActiveLots)

- Parameters

Boolean includeActiveLots - If True, include active lots or sublots in results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setIncludeActiveLots(True)
```

setIncludeInactiveLots(includeInactiveLots)

Description

If set to True, lots or sublot that are completed will be included in the results.

Syntax

setIncludeInactiveLots(includeInactiveLots)

- Parameters

Boolean includeActiveLots - If True, include completed lots or sublots in results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setIncludeInactiveLots(True)
```

setLotEquipmentClassFilter(lotEquipmentClassFilter)

Description

Set the lot equipment class filter to include lots that were stored in the equipment that belong to the equipment class that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setLotEquipmentClassFilter(lotEquipmentClassFilter)

- Parameters

String lotEquipmentClassFilter - The lot equipment class filter used to filter the results.

- Returns

Nothing

Code Examples

```
Code Snippet

#Example
filter = system.mes.lot.filter.createFilter()
filter.setLotEquipmentClassFilter('Storage Tank')
```

setLotEquipmentNameFilter(lotEquipmentNameFilter)

Description

Set the lot equipment name filter to include lots that were stored in the equipment with a name that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setLotEquipmentNameFilter(lotEquipmentNameFilter)

- Parameters

String lotEquipmentNameFilter - The lot equipment name filter used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setLotEquipmentNameFilter('Vinegar Tank?')
```

setLotNameFilter(lotNameFilter)

Description

Set the lot name to filter to include in the results. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setLotNameFilter(lotNameFilter)

- Parameters

String lotNameFilter - The lot name filter used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setLotNameFilter('V100*')
```

setLotStatusFilter(lotStatusFilter)

Description

Set the custom lot status of results to return. If the Final Lot Status property in a resource definition of a Process Segment or Operations Segment is set to a custom lot status, it can be filtered with this property.

Syntax

setLotStatusFilter(lotStatusFilter)

- Parameters

String lotStatusFilter - Custom lot status value to filter results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setLotStatusFilter('Complete')
```

setMaterialClassFilter(materialClassFilter)

Description

Set the material class filter to include lots that have material that belong to the material class that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setMaterialClassFilter(materialClassFilter)

- Parameters

String materialClassFilter - The material class filter used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setMaterialClassFilter('* Vinegar')
```

setMaterialNameFilter(materialNameFilter)

Description

Set the material definition name filter to include lots that have material with a name that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setMaterialNameFilter(materialNameFilter)

- Parameters

String materialNameFilter - The material definition name filter used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setMaterialNameFilter('* Turkey')
```

setMaxResults(maxResults)

Description

Set the maximum results to return. This prevents a large list from being returned which reduces database operations, memory usage and other resources when most of the time, the results are not used. If large results are needed, then this property can be increased.

Syntax

setMaxResults(maxResults)

- Parameters

Integer maxResults - The maximum number of items to return.

- Returns

Nothing

Code Examples

Code Snippet

```
#Here is an example of how to set the maximum result count.
filter = system.mes.lot.filter.createFilter()
filter.setMaxResults(200)
print filter.getMaxResults()
```

Output

```
200
```

setModeName(modeName)

Description

Set the type of results to return. It can be return results for lots (batches) of material or serialized items (sublots). Options are Lot and Sublot.

Syntax

setModeName(modeName)

- Parameters

String modeName - The name of the mode for the type of results to return.

- Returns

Nothing

Code Examples

Code Snippet

```
#This code snippet will set the mode name.
filter = system.mes.lot.filter.createFilter()
filter.setModeName('Sublot')
```

setOperationNameFilter(operationNameFilter)

Description

Set the operation name filter to include lots that were processed by the operation with a name that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setOperationNameFilter(operationNameFilter)

- Parameters

String operationNameFilter - The operation name filter used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setOperationNameFilter('Receive*')
```

setPersonnelClassFilter(personnelClassFilter)

Description

Set the personnel class filter to include lots that were processed by personnel that belong to the personnel class that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setPersonnelClassFilter(personnelClassFilter)

- Parameters

String personnelClassFilter - The personnel class filter used to filter the results.

- Returns

Nothing

Code Examples

```
Code Snippet

#Example
filter = system.mes.lot.filter.createFilter()
filter.setPersonnelClassFilter('Operator?')
```

setPersonnelNameFilter(personnelNameFilter)

Description

Set the personnel name filter to include lots that were processed with a name that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setPersonnelNameFilter(personnelNameFilter)

- Parameters

String personnelNameFilter - The personnel name filter used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setPersonnelNameFilter('Jo*')
```

setSegmentEquipmentClassFilter(segmentEquipmentClassFilter)

Description

Set the segment equipment class filter to include lots that were processed at the equipment that belong to the equipment class that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setSegmentEquipmentClassFilter(segmentEquipmentClassFilter)

- Parameters

String segmentEquipmentClassFilter - The segment equipment class filter used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setSegmentEquipmentClassFilter('* Tank')
```

setSegmentEquipmentNameFilter(segmentEquipmentNameFilter)

Description

Set the segment equipment name filter to include lots that were processed at the equipment with a name that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setSegmentEquipmentNameFilter(segmentEquipmentNameFilter)

- Parameters

String segmentEquipmentNameFilter - The segment equipment name filter used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setSegmentEquipmentNameFilter('Vinegar*')
```

setSegmentNameFilter(segmentNameFilter)

Description

Set the segment name filter to include lots that were processed by the segment with a name that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setSegmentNameFilter(segmentNameFilter)

- Parameters

String segmentNameFilter - The segment name filter used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setSegmentNameFilter('Receive*')
```

setSublotNameFilter(sublotNameFilter)

Description

Set the sublot name to filter to include in the results. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setSublotNameFilter(sublotNameFilter)

- Parameters

String sublotNameFilter - The sublot name filter used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setSublotNameFilter('BB 100?')
```

Properties:

getAvailable()

Description

Get the available quantity of the item. Available quantity = Quantity of material fed into the segment in the beginning - (Quantity fed out of the segment + scheduled quantity)

Syntax

getAvailable()

- Parameters

None

- Returns

Double The quantity of material that is available.

- Scope

All

getInQuantity()

Description

Get the quantity of material feeding into a segment that will be part of the finished goods.

Syntax

getInQuantity()

- Parameters

None

- Returns

Double The quantity of material.

- Scope

All

getLocationLink()

Description

Get the link to the location of the material lot

Syntax

getLocationLink()

- Parameters

None

- Returns

MES Object Link - The link representing the location.

- Scope

All

getLotNumber()

Description

Get the lot number of the material lot used to filter the results.

Syntax

getLotNumber()

- Parameters

None

- Returns

String materialLotNumber - The lot number to return the material lot object for.

- Scope

All

getLotSequence()

Description

Get the lot sequence of the material lot used to filter the results. The sequence property is incremented every time a new Material Lot is created for a given lot number.

Syntax

getLotSequence()

- Parameters

None

- Returns

Integer sequenceNumber - The lot sequence number to return the link for. If it is less than zero, then the lot holding the maximum value is returned and if the same lot number is used for multiple segments, each lot with the same lot number will be assigned a different lot sequence number.

- Scope

All

getMaterialDescription()

Description

The description about the material.

Syntax

getMaterialDescription()

- Parameters

None

- Returns

String The description of the specified material.

- Scope

All

getMaterialLot()

Description

Return the material lot MES object associated the specified material property of a segment.

Syntax

getMaterialLot()

- Parameters

None

- Returns

String - A MESMaterialLot object that is associated with the specified material property for segment.

- Scope

All

getMaterialLotUUID()

Description

Get the UUID of the material lot used to filter the results.

Syntax

getMaterialLotUUID()

- Parameters

None

- Returns

String materialLotUUID - The UUID of the Material Definition to assign to the Material Lot object.

- Scope

All

getMaterialName()

Description

The name of the material to assign to the Material Lot object.

Syntax

getMaterialName()

- Parameters

None

- Returns

String The name of the specified material.

- Scope

All

getMaterialUUID()

Description

The UUID of the material to assign to the Material Lot object. See UUIDs for more information.

Syntax

getMaterialUUID()

- Parameters

None

- Returns

String The UUID of the specified material.

- Scope

All

getNetQuantity()

Description

Get the net quantity of material in the lot. Net quantity = quantity in the beginning - used quantity of the item.

Syntax

getNetQuantity()

- Parameters

None

- Returns

Double The net quantity of material.

- Scope

All

getUnits()

Description

This specifies the units for the quantity setting.

Syntax

getUnits()

- Parameters

None

- Returns

String units - The units of quantity.

- Scope

All

setAvailable()

Description

Get the available quantity of the item. Available quantity = Quantity of material fed into the segment in the beginning - (Quantity fed out of the segment + scheduled quantity)

Syntax

setAvailable()

- Parameters

Double The quantity of material that is available.

- Returns

Nothing

- Scope

All

setInQuantity()

Description

Set the quantity of material feeding into a segment that will be part of the finished goods.

Syntax

setInQuantity()

- Parameters

Double The quantity of material.

- Returns

Nothing

- Scope

All

setLocationLink()

Description

Get the link to the location of the material lot

Syntax

setLocationLink()

- Parameters

MES Object Link - The link representing the location.

- Returns

Nothing

- Scope

All

setLotNumber()

Description

Set the lot number of the material lot used to filter the results.

Syntax

setLotNumber()

- Parameters

String materialLotNumber - The lot number to return the material lot object for.

- Returns

Nothing

- Scope

All

setLotSequence()

Description

Set the lot sequence of the material lot used to filter the results. The sequence property is incremented every time a new Material Lot is created for a given lot number.

Syntax

setLotSequence()

- Parameters

Integer sequenceNumber - The lot sequence number to return the link for. If it is less than zero, then the lot holding the maximum value is returned and if the same lot number is used for multiple segments, each lot with the same lot number will be assigned a different lot sequence number.

- Returns

Nothing

- Scope

All

setMaterialDescription()

Description

The description about the material.

Syntax

setMaterialDescription()

- Parameters

String The description of the specified material.

- Returns

Nothing

- Scope

All

setMaterialLot()

Description

Return the material lot MES object associated the specified material property of a segment.

Syntax

setMaterialLot()

- Parameters

String - A MESMaterialLot object that is associated with the specified material property for segment.

- Returns

Nothing

- Scope

All

setMaterialLotUUID()

Description

Set the UUID of the material lot used to filter the results.

Syntax

setMaterialLotUUID()

- Parameters

String materialLotUUID - The UUID of the Material Definition to assign to the Material Lot object.

- Returns

Nothing

- Scope

All

setMaterialName()

Description

The name of the material to assign to the Material Lot object.

Syntax

setMaterialName()

- Parameters

String The name of the specified material.

- Returns

Nothing

- Scope

All

setMaterialUUID()

Description

The UUID of the material to assign to the Material Lot object. See UUIDs for more information.

Syntax

setMaterialUUID()

- Parameters

String The UUID of the specified material.

- Returns

Nothing

- Scope

All

setNetQuantity()

Description

Get the net quantity of material in the lot. Net quantity = quantity in the beginning - used quantity of the item.

Syntax

setNetQuantity()

- Parameters

Double The net quantity of material.

- Returns

Nothing

- Scope

All

setUnits()

Description

This specifies the units for the quantity setting.

Syntax

setUnits()

- Parameters

String units - The units of quantity.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
equipLotSummary = system.mes.getLotInventoryByEquipment('[globa
l]\My Enterprise\California\Storage\Vinegar Tanks\Vinegar Tank
1')
for lotSummary in equipLotSummary:
    lotUUID = lotSummary.getMaterialLotUUID()
    lotNo = lotSummary.getLotNumber()
    lotSeq = lotSummary.getLotSequence()
    matUUID = lotSummary.getMaterialUUID()
    matName = lotSummary.getMaterialName()
    matDescription = lotSummary.getMaterialDescription()
```

```
    inQuant = lotSummary.getInQuantity()
    outQuant = lotSummary.getOutQuantity()
    schedule = lotSummary.getScheduled()
    available = lotSummary.getAvailable()
    netQuant = lotSummary.getNetQuantity()
    units = lotSummary.getUnits()
    locationLink = lotSummary.getLocationLink()
    print "lot uuid: %s, lot number: %s, lot sequence: %d,
material UUID: %s, material name: %s, material Description: %
s, In Quantity: %f, Out Quantity: %f, schedule: %f, available:
%f, net quantity: %f, units: %s, locationLink: %s" % (lotUUID,
lotNo, lotSeq, matUUID, matName, matDescription, inQuant,
outQuant, schedule, available, netQuant, units, locationLink)
```

Output

```
lot uuid: 5afdec7b-c546-4fcb-a5b2-399108966952, lot number: BB
1000, lot sequence: 1, material UUID: 8713506e-b179-4598-a324-
b21d5457a3a8, material name: Butterball Turkey, material
Description: , In Quantity: 0.000000, Out Quantity:
1000.000000, schedule: 0.000000, available: -1000.000000, net
quantity: -1000.000000, units: None, locationLink: Vinegar
Tank 1
lot uuid: 9ddeeb1b-23f5-43d6-b9db-c0567134aa12, lot number: BB
1002, lot sequence: 1, material UUID: 8713506e-b179-4598-a324-
b21d5457a3a8, material name: Butterball Turkey, material
Description: , In Quantity: 100.000000, Out Quantity:
0.000000, schedule: 0.000000, available: 100.000000, net
quantity: 100.000000, units: None, locationLink: Vinegar Tank 1
lot uuid: 5cd8b1e9-8238-490d-8cbb-108c3960aec6, lot number: BB
1003, lot sequence: 1, material UUID: 8713506e-b179-4598-a324-
b21d5457a3a8, material name: Butterball Turkey, material
Description: , In Quantity: 100.000000, Out Quantity:
0.000000, schedule: 0.000000, available: 100.000000, net
quantity: 100.000000, units: None, locationLink: Vinegar Tank 1
```

Properties:

getInQuantitySum()

Description

Get the quantity of a specific item inside the segment that will be part of the finished goods.

Syntax

getInQuantitySum()

- Parameters

None

- Returns

Double The quantity of a specific item inside the segment.

getNetQuantitySum()

Description

Get the net quantity of a specific item. Net Quantity = inQuantity - outQuantity.

Syntax

getNetQuantitySum()

- Parameters

None

- Returns

Double The net quantity of a specific item in the lot.

getOutQuantitySum()

Description

Get the quantity of a specific item moved out of the segment that is or will be part of the finished goods.

Syntax

getOutQuantitySum()

- Parameters

None

- Returns

Double The quantity of a specific item moved out of a segment.

getScheduledSum()

Description

Get the quantity of a specific item within a schedule.

Syntax

getScheduledSum()

- Parameters

None

- Returns

Double The quantity of a specific item within a schedule.

Code Examples

Code Snippet

```
#This simple example includes all the above mentioned
properties.
lotSummary = system.mes.getLotInventoryByEquipment('[global]
\My Enterprise\California\Storage\Vinegar Tanks\Vinegar Tank 1'
)
if lotSummary!= None:
    netQuant = lotSummary.getNetQuantitySum()
    inQuant = lotSummary.getInQuantitySum()
    outQuant = lotSummary.getOutQuantitySum()
    schedule = lotSummary.getScheduledSum()
    print " Net Quantity Sum: %f\n In Quantity Sum: %f\n Out
Quantity Sum: %f\n Scheduled Sum:%f" % (netQuant, inQuant,
outQuant, schedule)
```

Output

```
Net Quantity Sum: 802.000000
In Quantity Sum: 1802.000000
Out Quantity Sum: 1000.000000
Scheduled Sum:0.000000
```

MES Object Collection

A MESObjectCollection object is used by MES objects to hold parents and children. Normally, the parent and child script functions of the MES objects should be used, but this is provided as a reference to the MESObjectCollection object itself and provides some additional functionality.

Methods:

get(uuid)

Description

Returns the MES object link for the specified UUID. If the specified UUID does not exist, None will be returned.

Syntax

get(uuid)

- Parameters

None

- Returns

String mesObjectLink - The MES object link corresponding to the uuid.

Code Examples

Code Snippet

```
#Object link corresponding to the specified uuid is returned.
mesObject = system.mes.loadMESObject('Vinegar', 'MaterialClass'
)
if mesObject != None:
    childList = mesObject.getChildCollection()
    print childList.get('5acf3c9f-2789-44af-888f-fce08d9972a7')
```

Output

```
Red Wine Vinegar
```

getList()

Description

Returns a list of MES object links. Depending if getParentCollection() or getChildCollection() is called to get the MESObjectCollection object, it will contain MES object links that are parents or children.

Syntax

getList()

- Parameters

None

- Returns

String mesObjectLinkList - A list containing MES object links.

Code Examples

Code Snippet

```
#This example reads the child MES object links that belong to
the Vinegar Material Class.
mesObject = system.mes.loadMESObject('Vinegar', 'MaterialClass'
)
if mesObject != None:
    childList = mesObject.getChildCollection().getList()
    for child in childList:
        print child.getName()
```

Output

```
Balsamic Vinegar
Red Wine Vinegar
White Vinegar
```

isEmpty()

Description

Returns True if no MES object links exist in the collection.

Syntax

isEmpty()

- Parameters

None

- Returns

Boolean - True if there are no MES object links in the collection.

Code Examples

Code Snippet

```
#Prints False because there are three object links in the
collection.
mesObject = system.mes.loadMESObject('Vinegar', 'MaterialClass'
)
if mesObject != None:
    childList = mesObject.getChildCollection()
    print childList.isEmpty()
```

Output

```
False
```

size()

Description

Returns the number of MES object links in the collection. Depending if getParentCollection() or getChildCollection() is called to get the MESObjectCollection object, it will represent the number of parents or children.

Syntax

size()

- Parameters

None

- Returns

Integer - The number of MES object links in the collection.

Code Examples

Code Snippet

```
#In this example, Vinegar material class has got three
children. Therefore it prints 3.
mesObject = system.mes.loadMESObject('Vinegar', 'MaterialClass'
)
if mesObject != None:
    childList = mesObject.getChildCollection()
    print childList.size()
```

Output

3

Properties:

None

MES Object Event Parameters

Description

The **MESObjectEventParameters** object is used with the MESScriptEvent to pass parameters to the MES object event. The **MESObjectEventParameters** object holds name value pairs where the name is a string and value is any value type of object.

This object is typically used when executing user MES object events to allow passing values to the event script. The system.mes.object.parameters.create() script function can also be used to create a new instance of this object.

Methods:

get

Description

Get the value of a parameter by name.

Syntax

get(parameterName)

- Parameters

String parameterName - The name of the parameter to return the value for.

- Returns

Object - The type return matches the type when the parameter was added to the collection using the put() function. If the parameter name does not exist in the collection, None is returned.

- Scope

All

Code Examples

Code Snippet

```
#Create a new parameter collection instance.
params = system.mes.object.parameters.create()

#Add parameters to it.
params.put('Kind', 'Dressing')
params.put('Priority', 'High')

#Print the parameter values
print params.get('Kind')
print params.get('Priority')
print params.get('Type')
```

Output

```
Dressing
High
None
```

put

Description

Add a name value pair to the parameters collection.

Syntax

put(parameterName, value)

- Parameters

String parameterName - The name of the parameter to add to the parameters collection.

String value - The value of the parameter.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Create a new parameter collection instance.
params = system.mes.object.parameters.create()

#Add parameters to it.
params.put('Kind', 'Dressing')
params.put('Priority', 'High')
```

size

Description

Get the number of parameters in the parameter collection.

Syntax

size()

- Parameters

None

- Returns

Integer - description

- Scope

All

Code Examples

Code Snippet

```
#Create a new parameter collection instance.
params = system.mes.object.parameters.create()

#Add parameters to it.
params.put('Kind', 'Dressing')
params.put('Priority', 'High')

#Print the parameter values
print params.size()
```

Output

```
2
```

MES Object Filter

Base Object

The MES Object Filter is derived from the MESAbstractObject and inherits all the exposed properties, methods and events for that object.

Scripting Functions

- system.mes.object.filter.createFilter() can be used to create an **MESObjectFilter** object
- system.mes.object.filter.parseCustomPropertyValueFilter(filter) takes in an **MESObjectFilter** object as a parameter

Example: Creating an MES Object Filter

```
myMESObjectFilter = system.mes.object.filter.createFilter()
print myMESObjectFilter.getMESObjectTypeName()
```

Object Description

The **MESObjectFilter** object is used to specify the MES objects to return for various components such as the MES Object Selector and script functions.

Methods

Beside the common MESAbstractObject methods, the following methods exist for the MES Object Filter.

getCustomPropertyNamePattern()

Description

Get the custom property name pattern used to filter the results.

Syntax

getCustomPropertyNamePattern()

- Parameters

None

- Returns

String The custom property name pattern.

getCustomPropertyValueFilter()

Description

Get the list of MESPropertyValueFilter used to filter the results.

Syntax

getCustomPropertyValueFilter()

- Parameters

None

- Returns

List of MESPropertyValueFilter - The custom property value list.

getEnabledStateName()

Description

Get the enable state to filter the results.

Syntax

getEnabledStateName()

- Parameters

None

- Returns

String name - The name of the enable state.

getExcludedEquipmentPathList()

Description

Gets the list of equipment paths that are excluded.

Syntax

getExcludedEquipmentPathList()

- Parameters

None

- Returns

List<String> pathList - The list of equipment paths that are excluded.

getMESObjectNamePattern()

Description

Get the MES object name pattern used to filter the results.

Syntax

getMESObjectNamePattern()

- Parameters

None

- Returns

String The MES object name pattern.

getMESObjectTypeName()

Description

Return the MES object type name the filter is set for.

Syntax

getMESObjectTypeName()

- Parameters

None

- Returns

String The MES object type name.

getMESObjectTypes()

Description

Gets the MES object types associated with the specified MES object filter.

Syntax

getMESObjectTypes()

- Parameters

None

- Returns

MESObjectTypes - A list of mes object types associated with this filter.

- Scope

All

getMESObjectUUIDList()

Description

Returns a list of MES object UUIDs to return in the results.

Syntax

getMESObjectUUIDList()

- Parameters

None

- Returns

List of String - A list of MES object UUIDs.

Code Snippet

```
filter = system.mes.object.filter.createFilter()
#Gets the list of uuid.
list = filter.getMESObjectUUIDList()
Addind another item to the list.
list.add('5253ccae-47b4-4dc2-954f-900ffa8636eb')
```

getPrimaryClassFilter()

Description

Gets the primary class filter that has been set.

Syntax

getPrimaryClassFilter()

- Parameters

None

- Returns

The primary class filter which was previously defined to filter the results.

getPrimaryMESObjectPath()

Description

Gets the primary MES object path that was set to filter the results.

Syntax

getPrimaryMESObjectPath()

- Parameters

None

- Returns

The primary MES object path to filter the results.

getPrimaryMESObjectUUID()

Description

Get the UUID of the primary MES object to include in the results.

Syntax

getPrimaryMESObjectUUID()

- Parameters

None

- Returns

String The primary MES object UUID.

hasCustomPropertyNamePattern()

Description

Checks for the existence of a custom property name pattern.

Syntax

hasCustomPropertyNamePattern()

- Parameters

None

- Returns

boolean - True, if there exist a custom property name pattern and False otherwise.

- Scope

All

hasCustomPropertyValueFilter()

Description

Checks if there is a custom property value to filter the results.

Syntax

hasCustomPropertyValueFilter()

- Parameters

None

- Returns

boolean - True, if there exist a custom property value filter and False otherwise.

- Scope

All

hasMESObjectNamePattern()

Description

Checks if there is an MES object name pattern to filter the results.

Syntax

hasMESObjectNamePattern()

- Parameters

None

- Returns

boolean - True, if there exist an MES object name pattern and False otherwise.

- Scope

All

hasMESObjectTypes()

Description

Checks whether there is any MES object type name to filter the results.

Syntax

hasMESObjectTypes()

- Parameters

None

- Returns

boolean - True, if there exist any MES object type defined to filter the results and False otherwise.

- Scope

All

hasMESObjectUUIDs()

Description

Checks if there is MES object uuids to filter the results.

Syntax

hasMESObjectUUIDs()

- Parameters

None

- Returns

boolean - True, if there exist some uuids to filter the results.

hasPrimaryClassFilter()

Description

Checks if there is any primary class filter associated with this MES object filter.

Syntax

hasPrimaryClassFilter()

- Parameters

None

- Returns

boolean - True, if there exist a primary class filter and False otherwise.

hasPrimaryMESObjectPath()

Description

Checks whether there is a primary MES object path to filter the results.

Syntax

hasPrimaryMESObjectPath()

- Parameters

None

- Returns

boolean - True, if there exist a primary MES object path and False otherwise.

hasPrimaryMESObjectUUID()

Description

Checks for the existence of primary MES object uuid to filter the results.

Syntax

hasPrimaryMESObjectUUID()

- Parameters

None

- Returns

boolean - True, if there exist a primary MES object uuid and False otherwise.

isIncludeRelated()

Description

The results can be limited to only include the related items that is defined by this property that evaluates to true.

Syntax

isIncludeRelated()

- Parameters

None

- Returns

boolean - True if the MES object include related items and False otherwise.

isShowEquipmentPath()

Description

Checks whether the ShowEquipmentPath property is set to True.

Syntax

isShowEquipmentPath()

- Parameters

None

- Returns

boolean - True if the ShowEquipmentPath property is set to True and False otherwise.

setCustomPropertyNamePattern(customPropertyNamePattern)

Description

Set the custom property name pattern to filter the results. If a MES object contains a custom property that matches the custom property name pattern, then it will be included in the results.

Syntax

setCustomPropertyNamePattern(customPropertyNamePattern)

- Parameters

String customPropertyNamePattern - The custom property name pattern used to filter the results.

- Returns

Nothing

Code Examples

```
Code Snippet
```

```
filter = system.mes.object.filter.createFilter()
filter.setCustomPropertyNamePattern('Type')
```

setCustomPropertyValueFilter(customPropertyValueFilter)

Description

Set the custom property filter expressions to filter the results. If a custom property of a MES object matches an expression in this list, then it will be included in the results. Use system.mes.object.filter.parseCustomPropertyValueFilter() script function to create the list of MESPropertyValueFilter objects.

Syntax

setCustomPropertyValueFilter(customPropertyValueFilter)

- Parameters

List of MES Property Value Filter customPropertyValueFilter - The custom property value list to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Create a filter.
filter = system.mes.object.filter.createFilter()
#Parses the expression and returns a list of
MESPropertyValueFilter objects that are used in filters.
list = system.mes.object.filter.parseCustomPropertyValueFilter(
'pH > 5.0,Width = 2.5')
filter.setCustomPropertyValueFilter(list)
```

Code Examples

Code Snippet

```
filter = system.mes.object.filter.createFilter()
list = system.mes.object.filter.parseCustomPropertyValueFilter(
'Item Number=A12SIK')
filter.setCustomPropertyValueFilter(list)
list = system.mes.searchMESObjects(filter)
for ndx in range(list.size()):
    mesObject = list.get(ndx)
    print mesObject.getName()
```

Output

```
84001
```

setEnableStateName(name)

Description

Set the enable state to filter the results.

Options:
DISABLED
ENABLED
BOTH

Syntax

setEnableStateName(name)

- Parameters

String name - The name of the enable state.

- Returns

Nothing

Code Examples

Code Snippet

```
#Here's how to set the enable state.
filter = system.mes.object.filter.createFilter()
filter.setEnableStateName('Disabled')
```

setExcludedEquipmentPathList(excludePaths)

Description

Sets the list of equipment paths (separated by commas) to exclude.

Syntax

setExcludedEquipmentPathList(excludePaths)

- Parameters

List<String> excludePaths - The list of equipment paths (separated by commas) to be excluded.

- Returns

Nothing

setExcludedEquipmentPathList(excludeList)

Description

Sets the list of equipment paths to exclude.

Syntax

setExcludedEquipmentPathList(excludeList)

- Parameters

List<String> pathList - The list of equipment paths to be excluded.

- Returns

Nothing

setIncludeRelated(includeRelated)

Description

Sets the include related property to filter the results.

Syntax

setIncludeRelated(includeRelated)

- Parameters

boolean includeRelated - Set this to True, if results should only include related objects and set to False otherwise.

- Returns

Nothing

setMESObjectNamePattern(mesObjectNamePattern)

Description

Set the MES object name pattern to include in the results. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setMESObjectNamePattern(mesObjectNamePattern)

- Parameters

String mesObjectNamePattern - The MES object name pattern used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Create a filter.
filter = system.mes.object.filter.createFilter()
#Here is an example for setting the name pattern.
filter.setMESObjectNamePattern('*Turkey')
list = system.mes.searchMESObjects(filter)
for ndx in range(list.size()):
    mesObject = list.get(ndx)
    print mesObject.getMESObjectType().getDisplayName()
```

Output

```
Response Material Definition
Material Definition
Response Material Class
Response Material Class
Material Class
```

setMESObjectTypeName(mesObjectTypeNames)

Description

Set the MES object type name to filter the results.

Syntax

setMESObjectTypeName(mesObjectTypeNames)

- Parameters

String name - Name of the MES object type to limit the results. Options are: EquipmentClass, Equipment, Enterprise, Site, Area, Line, LineCell, LineCellGroup, StorageZone, StorageUnit.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
filter = system.mes.object.filter.createFilter()
#Name of MESObjectType is set to "EquipmentClass."
filter.setMESObjectTypeName('EquipmentClass')
```

setMESObjectTypes(mesObjectTypes)

Description

Sets the MES object types to filter the results.

Syntax

setMESObjectTypes(mesObjectTypes)

- Parameters

MESObjectTypes - The MES object types to set as filter.

- Returns

Nothing

- Scope

All

setMESObjectUUIDList(mesObjectUUIDList)

Description

Set the UUIDs of the MES objects to return in the results.

Syntax

setMESObjectUUIDList(mesObjectUUIDList)

- Parameters

List of String mesObjectUUIDList - The list of UUIDs to include in the results.

- Returns

Nothing

setPrimaryClassFilter(primaryClassFilter)

Description

The results can be limited to only include items that have a primary class filter defined by this property that evaluates to true.

Syntax

setPrimaryClassFilter(primaryClassFilter)

- Parameters

String primaryClassFilter - The primary class to filter the results.

- Returns

Nothing

setPrimaryMESObjectPath(primaryMESObjectPath)

Description

Set the path of the primary MES object to include in the results.

Syntax

setPrimaryMESObjectPath(primaryMESObjectPath)

- Parameters

String primaryMESObjectPath - The path of the primary MES object to include the results.

- Returns

Nothing

setPrimaryMESObjectUUID(primaryMESObjectUUID)

Description

Set the UUID of the primary MES object to include in the results. Child MES objects will also be included in the results.

Syntax

setPrimaryMESObjectUUID(primaryMESObjectUUID)

- Parameters

String primaryMESObjectUUID - The UUID of the primary MES object to include the results.

- Returns

Nothing

Code Examples

Code Snippet

```
filter = system.mes.object.filter.createFilter()
filter.setPrimaryMESObjectUUID('73facb39-806c-4bfc-8881-
cc06707a9909')
```

setShowEquipmentPath(showPath)

Description

If set to True the equipment names with the paths are shown.

Syntax

setShowEquipmentPath(showPath)

- Parameters

Boolean showPath - Set to True to display equipment path and False otherwise.

- Returns

Nothing

Properties

The following properties are available for this object.

- None

MES Object Link

Object Description

The MESObjectLink object is a light weight version of an MES object. Think of it as a reference to the full MES object. In many cases, just the general information about an MES object is needed instead of the full details of the MES object.

Consider the case of displaying options in the MES Object Selector component. For example, there may be 100 different names populating the drop down selector. Loading the full details of each MES object loads a lot of unnecessary data and creates extra overhead. Instead a reference to each item in the MES Object Selector could be loaded with a link that includes the name, UUID, and type for each MES object. This approach is faster for the operator and less costly to the server.

Scripting Functions

The following functions can be used to create an **MESObjectLink** object.

- system.mes.object.link.create(mesObject)
- system.mes.object.link.create(mesObjectType, mesObjectUUID)
- system.mes.object.link.create(mesObjectType, mesObjectUUID, name)
- system.mes.object.link.create(mesObjectTypeName, mesObjectUUID)
- system.mes.object.link.create(mesObjectTypeName, mesObjectUUID, name)

Example

From the MESObjectLink object the full MES object can be loaded by calling the getMESObject() method.

Code Snippet

```
mesObject = mesObjectLink.getMESObject()
```

Methods

The following methods exist for the MES Object Link.

getMESObject()

Description

Returns MES object associated with a MESObjectLink object. If the MES object has not been previously loaded, it will be loaded.

Syntax

getMESObject()

- Parameters

None

- Returns

AbstractMESObject - The MES object associated with the event.

Code Examples

Code Snippet

```
#Example
objLink = system.mes.object.link.create('ProcessSegment', '4c8e
dc76-d08b-4838-9e3f-562151eecfa0')
print objLink.getMESObject()
```

Output

```
ProcessSegment (4c8edc76-d08b-4838-9e3f-562151eecfa0, Receive
Steel, 1 parents, 0 children, 1 custom properties, 4 complex
properties)
```

getMESObjectType()

Description

Returns MES object type that this MESObjectLink object is set to.

Syntax

getMESObjectType()

- Parameters

None

- Returns

MESObjectTypes

Code Examples

Code Snippet

```
#Example
obj = system.mes.loadMESObject('57290b1b-c734-404a-bc0f-
af0f5e4d2a91')
objLink = system.mes.object.link.create(obj)
print objLink.getMESObjectType()
```

Output

```
Material Class
```

getMESObjectUUID()

Description

Returns MESObject UUID that this MESObjectLink object is set to.

Syntax

getMESObjectUUID()

- Parameters

None

- Returns

String

Code Examples

Code Snippet

```
#Example
matCls = system.mes.loadMESObject('Turkey', 'MaterialClass')
objLink = system.mes.object.link.create(matCls)
print objLink.getMESObjectUUID()
```

Output

```
57290b1b-c734-404a-bc0f-af0f5e4d2a91
```

getName()

Syntax

getName()

- Parameters

None

- Returns

String

Code Examples

Code Snippet

```
#Example
obj = system.mes.loadMESObject('8da06ff8-2922-4e0c-a01a-
e7cda6899a0e')
objLink = system.mes.object.link.create(obj)
print objLink.getName()
```

Output

```
Vinegar Tank 1
```

hasMESObject()

Description

Returns True if MES object has been loaded into this MESObjectLink. This will be the case either after the system.mes.create(mesObject) constructor or the getMESObject() property of a MESObjectLink object has been called.

Syntax

hasMESObject()

- Parameters

None

- Returns

Boolean

Code Examples

Code Snippet

```
#Example
obj = system.mes.loadMESObjectByEquipmentPath('[global]\My
Enterprise\California\Receiving\Unload Station 1')
objLink = system.mes.object.link.create(obj)
print objLink.hasMESObject()
```

Output

```
True
```

hasMESObjectInfo()

Description

Returns True if MES object information exists. This includes the MES object type and UUID. The name is not required for this property to return True.

Syntax

hasMESObjectInfo()

- Parameters

None

- Returns

Boolean

Code Examples

Code Snippet

```
#Example
objLink = system.mes.object.link.create('MaterialDef', 'Box')
print objLink.hasMESObjectInfo()()
```

Output

```
True
```

hasName()

Description

Returns True if the MES object name is available.

Syntax

hasName()

- Parameters

None

- Returns

Boolean

Code Examples

Code Snippet

```
#Example
obj = system.mes.loadMESObjectByEquipmentPath('[global]\My
Enterprise\California\Storage\Vinegar Tanks\Vinegar Tank 1')
objLink = system.mes.object.link.create(obj)
print objLink.hasName()
```

Output

```
True
```

invalidateMESObject()

Description

This clears the previously loaded MES object associated with this MESObjectLink. This can be used to reuse a MESObjectLink object.

Syntax

invalidateMESObject()

- Parameters

None

- Returns

Nothing

Code Examples

Code Snippet

```
obj = system.mes.loadMESObject('Red Wine', 'MaterialDef')
objLink = system.mes.object.link.create(obj)
objLink.invalidateMESObject()
print objLink.hasMESObject()
```

Output

```
False
```

MES Object List

Object Description

The **MESObjectList** is a collection of MES objects. A list may contain any number of mes objects. From the MESObjectList object an MES object with a specific uuid can be loaded by calling the findByUUID() function.

Scripting Functions

The following function can be used to create an **MESObjectList** object.

system.mes.object.list.createList()

Description

This script function is used to create a list of MES objects.

Syntax

system.mes.object.list.createList()

- Parameters

None

- Returns

The list of MES objects.

- Scope

All

Example

```
obj = system.mes.loadMESObject('Box', 'MaterialDef')
objList = system.mes.object.list.createList()
objList.add(obj)
```

Methods

The following methods exist for the MES Object List.

add(mesobject)

Description

This script function is used to add a MES object to the list.

Syntax

add(mesobject)

- Parameters

AbstractMESObject mesobject - The object to be added.

- Returns

True if the object is added and False otherwise.

- Scope

All

remove(mesobject)

Description

This script function is used to remove a MES object to the list.

Syntax

remove(mesobject)

- Parameters

AbstractMESObject mesobject - The object to be removed.

- Returns

True if the object is removed and False otherwise.

- Scope

All

addAll(collection)

Description

Appends all of the elements in the specified collection to the end of this MES object list.

Syntax

addAll(collection)

- Parameters

List collection - A collection containing elements to be added to this list.

- Returns

True if all the objects are added to this MES object list and False otherwise.

- Scope

All

removeAll(collection)

Description

Removes all of the elements from the MES object list.

Syntax

removeAll(collection)

- Parameters

List collection - A collection containing elements to be removed to this list.

- Returns

True if the objects in the list is removed and False otherwise.

- Scope

All

findByUUID()

Description

Find a specific object from a list of MES objects by UUID.

Syntax

findByUUID(uuid)

- Parameters

String uuid - UUID of the MES object.

- Returns

AbstractMESObject mesObject - The MES object corresponding to the specific UUID.

- Scope

All

hasSingleMESObject()

Description

Checks whether the list contains more than one MES object.

Syntax

hasSingleMESObject()

- Parameters

None

- Returns

Boolean

- Scope

All

Code Example

Code Snippet

```
#Creates a list of MES objects
objList = system.mes.object.list.createList()

#Load the objects to be added
m1 = system.mes.loadMESObject('Bulk Almonds', 'MaterialDef')
m2 = system.mes.loadMESObject('Bulk Peanuts', 'MaterialDef')

#Adds the objects to list
objList.add(m1)
objList.add(m2)
```

```
#Save the changes
system.mes.saveMESObjects(objList)

#This code snippet will check if the list contains only one
MES object
print objList.hasSingleMESObject()

#Gets info about the object specified by the uuid
print objList.findByUUID('a3f05165-1cee-4661-a1e8-d282bf2c6a02'
)

#Creates a filter
filter = system.mes.object.filter.createFilter()
filter.setEnableStateName('ENABLED')
filter.setMESObjectNamePattern('Receive *')
mesList = system.mes.loadMESObjects(filter)

#Adds the elements in list 'mesList' to the list 'objList'
objList.addAll(mesList)

#Removes all the objects from the list
objList.removeAll(mesList)
```

Output

```
True
True
False
MaterialDef (a3f05165-1cee-4661-a1e8-d282bf2c6a02, Bulk
Almonds, 1 parents, 0 children, 2 custom properties, 0 complex
properties)
True
True
```

MES Object Type Name

The MES Object Type Name is a string parameter that is passed or returned by a number of MES objects and system.mes scripting functions.

ResponsePersonnelClass
ResponsePerson
PersonnelClass
Person
ResponseMaterialClass

ResponseMaterialDef
MaterialLot
MaterialSublot
MaterialRoot
MaterialClass
MaterialDef
ResponseEquipmentClass
ResponseEquipment
MESResponseEnterprise
MESResponseSite
MESResponseArea
MESResponseLine
MESResponseLineCell
MESResponseLineCellGroup
MESResponseStorageZone
MESResponseStorageUnit
EquipmentClass
Equipment
MESEnterprise
MESSite
MESArea
MESLine
MESLineCell
MESLineCellGroup
MESStorageZone
MESStorageUnit
ProcessSegment
OperationsDefinition
OperationsSegment
OperationsVersion
VersionSegment
OperationsSchedule
OperationsRequest
RequestSegment
OperationsPerformance
OperationsResponse
ResponseSegment
EquipmentState
EquipmentStateClass
EquipmentStateRoot
EquipmentMode

EquipmentModeClass
EquipmentModeRoot
AnalysisSettings
WorkOrder

MESObjectTypes

The MESObjectTypes object has some helpful functions when working with MES Objects (T&T) . Both the AbstractMESObject and the MES Object Link object s have a getMESObjectType() function that returns the type of an MES object .

The code below shows how to determine the specific MES object type using the getMESObjectType() method on the object.

Get MES Object Type

```
##This code snippet will print the names of MES object types
##for the objects returned by the filter.
##For this example there is an Equipment Class called "Packaging
Equipment".

filter = system.mes.object.filter.createFilter()
filter.setMESObjectNamePattern('Packaging Equipment')
list = system.mes.searchMESObjects(filter)
for ndx in range(list.size()):
    mesObject = list.get(ndx)
    print mesObject.getMESObjectType().getDisplayName()
```

Result

```
Equipment Class
```

Methods

getDescription()

Description

Returns description for the MES object type.

Syntax

getDescription()

- Parameters

None

- Returns

String description - The description of the MES object type.

- Scope

All

getDisplayName()

Description

Return the display name of the MES object type. This will be the long name with spaces.

Syntax

getDisplayName()

- Parameters

None

- Returns

String name - The display name of the MES object type.

- Scope

All

getName()

Description

Returns the name of the MES object type. This will be a short name without spaces.

Syntax

getName()

- Parameters

None

- Returns

String name - The name of the MES object type.

- Scope

All

getTypeFromName()

Description

Returns a MES Object Types object for the specified type name.

Syntax

getTypeFromName(name)

- Parameters

String name - The short name of the MES object type.

- Returns

The MESObjectTypes object.

- Scope

All

MES Property Value Filter

The MESPropertyValueFilter object is used to define custom property value filters that are used by script functions.

Methods:

system.mes.object.filter.parseCustomPropertyValueFilter(filterExpression)

Description

Parses a string expression and returns a list of MESPropertyValueFilter objects that are used in filters. The expression consists of the custom property path, a comparison and a constant value.

The custom property path starts with the MES object types and continues with the route including nested custom properties to the final custom property to be used in the comparison.

Example Paths:

MaterialLot.pH
MaterialLot.Dimension.Width

The path is followed by a comparison that can be any one of the following:

>	Greater than
>=	Greater than equal
<	Less than
<=	Less than equal
=	Equal
!=	Not equal

Multiple expressions can be included by separating then with commas. When this is done, all sub expressions must match for the overall expression to evaluate to true.

Example Expressions:

MaterialLot.pH >= 4.5
MaterialLot.Dimension.Width = 5
ResponseSegment.BreadType = Wheat
MaterialLot.Dimension.Depth >= 6.2,MaterialLot.Dimension.Width = 5

Syntax

- Parameters

String filterExpression - The custom property filter expression to parse.

- Returns

Based on the filterExpression, a list of MESPropertyValueFilter objects.

Code Snippet

```
filter = system.mes.object.filter.createFilter()
list = system.mes.object.filter.parseCustomPropertyValueFilter('pH
> 5.0,Width = 2.5')
filter.setCustomPropertyValueFilter(list)
results = system.mes.loadMESObjects(filter)
for link in results:
    print link.getName()
```

Properties:

None

MES Response Material Property

getAllocatedQuantity()

Description

Gets the allocated quantity as double.

Syntax

getAllocatedQuantity()

- Parameters

None

- Returns

Double value - The allocated quantity of the lot.

- Scope

All

getAutoGenerateLot()

Description

Gets a boolean indicating whether this response material property will automatically generate lots or not.

Syntax

getAutoGenerateLot()

- Parameters

None

- Returns

boolean autoGenerateLot - True, if the autoGenerateLot property is set to true and False otherwise.

- Scope

All

getAutoLotCompletion()

Description

Gets the value of auto lot completion property associated with this response material property.

Syntax

getAutoLotCompletion()

- Parameters

None

- Returns

The value set for the auto lot completion property.

- Scope

All

getBeginDateTime()

Description

Get the beginning date and time to limit the results to return.

Syntax

getBeginDateTime()

- Parameters

None

- Returns

Date beginDateTime - The date and time to filter the results.

- Scope

All

getCycleTime()

Description

Gets the lot cycle time in seconds.

Syntax

getCycleTime()

- Parameters

None

- Returns

Integer value - The lot cycle time in seconds.

- Scope

All

getDefaultQuantity()

Description

Gets the default quantity of the ResponseMaterial property.

Syntax

getDefaultQuantity()

- Parameters

None

- Returns

Double value - The default quantity value.

- Scope

All

getEnableSublots()

Description

A boolean indicating whether this response material property will enable sublots or not.

Syntax

getEnableSublots()

- Parameters

None

- Returns

boolean enableSublots - True, if sublots are enabled and False otherwise.

- Scope

All

getEndDateTime()

Description

Get the end date and time to limit the results to return.

Syntax

getEndDateTime()

- Parameters

None

- Returns

Date endDateTime - The date and time to filter the results.

- Scope

All

getEquipmentRef()

Description

Returns the reference of the equipment property.

Syntax

getEquipmentRef()

- Parameters

None

- Returns

MES Object Link mesObjectLink - The MESObjectLink representing the equipment.

- Scope

All

getEquipmentRefProperty()

Description

Gets the equipment reference properties for this response material property.

Syntax

getEquipmentRefProperty()

- Parameters

None

- Returns

MESLotEquipmentRefProperty - The equipment reference properties. All those children properties to the MES response material property referring the equipment.

- Scope

All

getEquipmentRefType()

Description

Gets the reference type of the equipment.

Syntax

getEquipmentRefType()

- Parameters

None

- Returns

String equipmentRefType - The type of the equipment.

- Scope

All

getEquipmentRefUUID()

Description

Gets the reference uuid of the equipment.

Syntax

getEquipmentRefUUID()

- Parameters

None

- Returns

String equipmentRefUUID - The uuid representing this equipment.

- Scope

All

getEquipmentVersionRefUUID()

Description

Gets the uuid representing the equipment version.

Syntax

getEquipmentVersionRefUUID()

- Parameters

None

- Returns

String getVersionRefUUID - The uuid to refer the version of the equipment.

- Scope

All

getFinalLotStatus()

Description

Gets the final lot status of the response material.

Syntax

getFinalLotStatus()

- Parameters

None

- Returns

String finalLotStatus - The final status of the material lot.

- Scope

All

getLotBeginningQuantity()

Description

Gets the quantity set to begin the lot.

Syntax

getLotBeginningQuantity()

- Parameters

None

- Returns

Double value - The quantity of material set at the beginning.

- Scope

All

getLotCycleTime()

Description

Gets the lot cycle time in seconds.

Syntax

getLotCycleTime()

- Parameters

None

- Returns

Integer cycleTime - Lot cycle time for this response material property.

- Scope

All

getLotDepletionSeconds()

Description

Gets the depletion time of lot in seconds.

Syntax

getLotDepletionSeconds()

- Parameters

None

- Returns

Integer lotDepletionSeconds - The depletion time of the lot.

- Scope

All

getLotDepletionWarning()

Description

Gets the warning of lot depletion in seconds.

Syntax

getLotDepletionWarning()

- Parameters

None

- Returns

Integer value - The depletion warning in seconds.

- Scope

All

getLotMessageType()

Description

Gets the message type of the material lot.

Syntax

getLotMessageType()

- Parameters

None

- Returns

MESLotMessageTypes lotMessageType - The lot message type.

- Scope

All

getLotNumberSource()

Description

Gets name of the lot number source.

Syntax

getLotNumberSource()

- Parameters

None

- Returns

String lotNoSource - The name of the lot number source.

- Scope

All

getLotNumberSourceLink()

Description

Gets the name of the lot number source link.

Syntax

getLotNumberSourceLink()

- Parameters

None

- Returns

String lotNoSourceLink - Name of the lot number source link.

- Scope

All

getLotRate()

Description

Gets the lot rate for this response material property.

Syntax

getLotRate()

- Parameters

None

- Returns

Double lotRate - The lot rate for this response material property.

- Scope

All

getLotRatePeriod()

Description

Gets the lot rate period for this response material property.

Syntax

getLotRatePeriod()

- Parameters

None

- Returns

String lotRatePeriod - The lot rate period for this response material property.

- Scope

All

getLotRefSequence()

Description

Gets the sequence number corresponding to the lot.

Syntax

getLotRefSequence()

- Parameters

None

- Returns

Integer value - The sequence number associated with the lot.

- Scope

All

getLotStatusFilter()

Description

Get the custom lot status of results to return.

Syntax

getLotStatusFilter()

- Parameters

None

- Returns

String lotStatusFilter - The custom lot status value.

- Scope

All

getLotUUID()

Description

Gets the uuid corresponding to this material lot.

Syntax

getLotUUID()

- Parameters

None

- Returns

String uuid - The unique identifier for this lot.

- Scope

All

getManualLotNo()

Description

Gets the manually entered lot number.

Syntax

getManualLotNo()

- Parameters

None

- Returns

String manualLotNum - The lot number entered by the user.

- Scope

All

getMaterialLot()

Description

Gets the material lot associated with this response material property.

Syntax

getMaterialLot()

- Parameters

None

- Returns

MESMaterialLot materialLot - The material lot object.

- Scope

All

getMaterialRef()

Description

Gets the MES object link corresponding to this response material property.

Syntax

getMaterialRef()

- Parameters

None

- Returns

MES Object Link mesObjectLink - The link corresponding to the material.

- Scope

All

getMaterialRefProperty()

Description

Gets the material reference properties for this response material property.

Syntax

getMaterialRefProperty()

- Parameters

None

- Returns

The material reference properties. All those children properties to MES response material property.

- Scope

All

getMaterialRefType()

Description

Gets the reference type of the material.

Syntax

getMaterialRefType()

- Parameters

None

- Returns

S tring materialRefType - The type of the material reference.

- Scope

All

getMaterialRefUUID()

Description

Gets the uuid for the material reference.

Syntax

getMaterialRefUUID()

- Parameters

None

- Returns

S tring materialRefUUID - The unique identifier for the material reference.

- Scope

All

getMaterialVersionRefUUID()

Description

Gets the version reference uuid for the material.

Syntax

getMaterialVersionRefUUID()

- Parameters

None

- Returns

String uuid - The uuid which represent the version of the material.

- Scope

All

getQuantity()

Description

Gets the quantity set for the lot.

Syntax

getQuantity()

- Parameters

None

- Returns

Double quantity - The actual quantity for this lot resource. This can be the current quantity at anytime during the life of a Response Segment, but will equal the final production quantity when this lot resource is finalized.

- Scope

All

getQuantitySource()

Description

This setting determines the source of the quantity for this response material resource.

Syntax

getQuantitySource()

- Parameters

None

- Returns

String quantitySource - The name of the source of the quantity.

- Scope

All

getQuantitySourceLink()

Description

Gets the name of the material resource to link to this segment. This is used when the Quantity Source setting is set to Link, Split or Combine.

Syntax

getQuantitySourceLink()

- Parameters

None

- Returns

String quantitySourceLink - The link to the quantity source.

- Scope

All

getQuantityUnits()

Description

Gets the quantity units for this response segment.

Syntax

getQuantityUnits()

- Parameters

None

- Returns

String quantityUnits - The units of lot quantity.

- Scope

All

getRate()

Description

Gets the lot rate for this response segment.

Syntax

getRate()

- Parameters

None

- Returns

Double rate - The lot rate set for this response segment object.

- Scope

All

getRatePeriod()

Description

Gets the material rate period.

Syntax

getRatePeriod()

- Parameters

None

- Returns

String ratePeriod - The material rate period.

- Scope

All

getReferenceOptions(mesObjectType, searchPattern)

Description

Get a list of the available options for the specified MES reference property specified by the object type and search pattern.

Syntax

getReferenceOptions(mesObjectType, searchPattern)

- Parameters

MESObjectTypes mesObjectType - The MES object type of the links that represent the options.

String searchPattern - The search pattern to filter the results by. It can contain the * and ? wild card characters.

- Returns

A list of MES Object Link objects holding the options that are appropriate for the specified property. The list is returned as a MES List object that is a collection holding MES object links that represent the options.

- Scope

All

getReferenceOptions(mesObjectType, searchPattern, maxLotReturnCount, lotDescriptionPattern)

Description

Get a list of the available options for the specified MES reference property specified by the parameters.

Syntax

getReferenceOptions(mesObjectType, searchPattern, maxLotReturnCount, lotDescriptionPattern)

- Parameters

MESObjectTypes mesObjectType - The MES object type of the links that represent the options.

String searchPattern - The search pattern to filter the results by. It can contain the * and ? wild card characters.

int maxLotReturnCount - The maximum number of lots.

String lotDescriptionPattern - This is a pattern to filter the lot descriptions.

- Returns

A list of MES Object Link objects holding the options that are appropriate for the specified property. The list is returned as a MES List object that is a collection holding MES object links that represent the options.

- Scope

All

getSegmentRefUUID()

Description

Gets the reference uuid of response segment.

Syntax

getSegmentRefUUID()

- Parameters

None

- Returns

String uuid - The reference uuid for the response segment.

- Scope

All

getStatus()

Description

Gets the custom lot status.

Syntax

getStatus()

- Parameters

None

- Returns

String status - The previously set lot status.

- Scope

All

getUnits()

Description

Gets the units set to the quantity settings.

Syntax

getUnits()

- Parameters

None

- Returns

String units - The units of quantity.

- Scope

All

getUse()

Description

Gets the material use types. Options are In, Out, Consumable, By-product.

Syntax

getUse()

- Parameters

None

- Returns

String lotUse - The lot use type for the response material.

- Scope

All

getZeroLotThresholdQty()

Description

Gets the value of the zero lot threshold quantity.

Syntax

getZeroLotThresholdQty()

- Parameters

None

- Returns

Double value - The zero lot threshold quantity value.

- Scope

All

hasBeginDateTime()

Description

Checks if the start date and time is set.

Syntax

hasBeginDateTime()

- Parameters

None

- Returns

boolean time - True, if the date and time to begin is set and False otherwise.

- Scope

All

hasEndDateTime()

Description

Checks whether the end date and time is set.

Syntax

hasEndDateTime()

- Parameters

None

- Returns

boolean time - True, if the end date and time is set and False otherwise.

- Scope

All

hasEquipmentRef()

Description

Checks for the existence of equipment reference.

Syntax

hasEquipmentRef()

- Parameters

None

- Returns

boolean True, if there exist any reference to the equipment and False otherwise.

- Scope

All

hasLotRef()

Description

Checks for the existence of lot reference.

Syntax

hasLotRef()

- Parameters

None

- Returns

boolean True, if there exist any reference to the lot and False otherwise.

- Scope

All

hasMaterialRef()

Description

Checks whether there is any reference to the material.

Syntax

hasMaterialRef()

- Parameters

None

- Returns

boolean True, if there is material reference and False otherwise.

- Scope

All

hasQuantity()

Description

Checks whether the quantity is set.

Syntax

hasQuantity()

- Parameters

None

- Returns

boolean True, if the quantity is specified for the response segment and False otherwise.

- Scope

All

hasQuantityUnits()

Description

Checks if the quantity units is set or not.

Syntax

hasQuantityUnits()

- Parameters

None

- Returns

boolean True, if there exist a quantity unit setting and False otherwise.

- Scope

All

hasSegmentRefUUID()

Description

Checks for a segment reference uuid.

Syntax

hasSegmentRefUUID()

- Parameters

None

- Returns

boolean True if there is a reference uuid for the segment and False otherwise.

- Scope

All

isActive()

Description

Checks if the lot is active.

Syntax

isActive()

- Parameters

None

- Returns

boolean - True if the lot is active and False if it is inactive.

- Scope

All

isEquipmentReadyToRun()

Description

Checks if the equipment reference is a valid type and ready to run.

Syntax

isEquipmentReadyToRun()

- Parameters

None

- Returns

boolean True if the equipment is ready to run and False otherwise.

- Scope

All

isMaterialReadyToRun()

Description

Checks if the material reference is a valid type.

Syntax

isMaterialReadyToRun()

- Parameters

None

- Returns

boolean True if the material reference type is valid and False otherwise.

- Scope

All

isOptional()

Description

Returns true if the lot is set as optional.

Syntax

isOptional()

- Parameters

None

- Returns

boolean True if the lot is set as optional and False otherwise.

- Scope

All

isOutUse()

Description

Checks if the material use type is out.

Syntax

isOutUse()

- Parameters

None

- Returns

boolean True if the material use type is out and False otherwise.

- Scope

All

isProductionSelectable()

Description

Checks if the production is selectable or not.

Syntax

isProductionSelectable()

- Parameters

None

- Returns

boolean True if the production is selectable and False otherwise.

- Scope

All

setAllocatedQuantity(allocatedQuantity)

Description

Sets the allocated quantity in double.

Syntax

setAllocatedQuantity(allocatedQuantity)

- Parameters

Double allocatedQuantity - The quantity to allocate the lot for.

- Returns

Nothing

- Scope

All

setAutoGenerateLot(autoGenerateLot)

Description

Sets a boolean indicating whether this response material property will automatically generate lots or not.

Syntax

setAutoGenerateLot(autoGenerateLot)

- Parameters

boolean autoGenerateLot - True, if the autoGenerateLot property is set to true and False otherwise.

- Returns

Nothing

- Scope

All

setAutoLotCompletion(autoLotCompletion)

Description

Sets the value of auto lot completion property associated with this response material property.

Syntax

setAutoLotCompletion(autoLotCompletion)

- Parameters

String autoLotCompletion - The value to set the auto lot completion property for.

- Returns

Nothing

- Scope

All

setBeginDateTime(beginDateTime)

Description

Set the beginning date and time to limit the results to return.

Syntax

setBeginDateTime(beginDateTime)

- Parameters

Date beginDateTime - The date and time to filter the results.

- Returns

Nothing

- Scope

All

setCycleTime(cycleTime)

Description

Sets the lot cycle time in seconds.

Syntax

setCycleTime(cycleTime)

- Parameters

Sets the lot cycle time in seconds.

- Returns

Nothing

- Scope

All

setDefaultQuantity(defaultQuantity)

Description

Sets the default quantity of the ResponseMaterial property.

Syntax

setDefaultQuantity(defaultQuantity)

- Parameters

Double defaultQuantity - The default quantity value.

- Returns

Nothing

- Scope

All

setEnableSublots(enableSublots)

Description

A boolean indicating whether this response material property will enable sublots or not.

Syntax

setEnableSublots(enableSublots)

- Parameters

boolean enableSublots - True, if sublots are enabled and False otherwise.

- Returns

Nothing

- Scope

All

setEndDateTime(endDateTime)

Description

Set the end date and time to limit the results to return.

Syntax

setEndDateTime(endDateTime)

- Parameters

Date endDateTime - The date and time to filter the results.

- Returns

Nothing

- Scope

All

setEquipmentRef(mesObjectLink)

Description

Sets the reference of the equipment.

Syntax

setEquipmentRef(mesObjectLink)

- Parameters

MES Object Link mesObjectLink - The MESObjectLink representing the equipment.

- Returns

Nothing

- Scope

All

setEquipmentRefType(equipmentRefType)

Description

Sets the reference type of the equipment.

Syntax

setEquipmentRefType(equipmentRefType)

- Parameters

String equipmentRefType - The type of the equipment.

- Returns

Nothing

- Scope

All

setEquipmentRefUUID(equipmentRefUUID)

Description

Sets the reference uuid of the equipment.

Syntax

setEquipmentRefUUID(equipmentRefUUID)

- Parameters

String equipmentRefUUID - The uuid representing this equipment.

- Returns

Nothing

- Scope

All

setFinalLotStatus(finalLotSttus)

Description

This is useful for setting a lot to Hold, In Process or anything that can be used to filter lots or sublots. When a segment is started, the status of the Material Lots will be set to Active. When the segment is ended or a new lot is used for the material resource, the status will be set to Complete. Optionally, the value of this setting can be used instead of the default Complete. Please note, the Active status while the lot is active cannot be changed.

Syntax

setFinalLotStatus(finalLotStatus)

- Parameters

String finalLotStatus - The status to set the lot for.

- Returns

Nothing

- Scope

All

setLotCycleTime(cycleTime)

Description

Sets the lot cycle time in seconds for the response material property.

Syntax

setLotCycleTime(cycleTime)

- Parameters

Integer cycleTime - Lot cycle time to set the property for.

- Returns

Nothing

- Scope

All

setLotDepletionSeconds(lotDepletionSeconds)

Description

Sets the depletion time of lot in seconds.

Syntax

setLotDepletionSeconds(lotDepletionSeconds)

- Parameters

Integer lotDepletionSeconds - The depletion time of the lot.

- Returns

Nothing

- Scope

All

setLotDepletionWarning(lotDepletionWarningSeconds)

Description

Sets the warning of lot depletion in seconds.

Syntax

setLotDepletionWarning(lotDepletionWarningSeconds)

- Parameters

Integer lotDepletionWarningSeconds - The depletion warning in seconds.

- Returns

Nothing

- Scope

All

setLotMessageType(lotMessageType)

Description

Sets the message type of the material lot.

Syntax

setLotMessageType(lotMessageType)

- Parameters

MESLotMessageTypes lotMessageType - The lot message type.

- Returns

Nothing

- Scope

All

setLotNetQuantity(lotNetQuantity)

Description

Sets the lot net quantity at the beginning.

Syntax

setLotNetQuantity(lotNetQuantity)

- Parameters

Double lotNetQuantity - The net quantity in double to set at the beginning.

- Returns

Nothing

- Scope

All

setLotNumberSource(lotNoSource)

Description

Sets name of the lot number source.

Syntax

setLotNumberSource(lotNoSource)

- Parameters

String lotNoSource - The name of the lot number source.

- Returns

Nothing

- Scope

All

setLotNumberSourceLink(lotNoSourceLink)

Description

Sets the name of the lot number source link.

Syntax

setLotNumberSourceLink(lotNoSourceLink)

- Parameters

String lotNoSourceLink - Name of the lot number source link.

- Returns

Nothing

- Scope

All

setLotRate(lotRate)

Description

Sets the lot rate for this response material property.

Syntax

setLotRate(lotRate)

- Parameters

Double lotRate - The lot rate for this response material property.

- Returns

Nothing

- Scope

All

setLotRatePeriod(lotRatePeriod)

Description

Sets the lot rate period for this response material property.

Syntax

setLotRatePeriod(lotRatePeriod)

- Parameters

String lotRatePeriod - The lot rate period for this response material property.

- Returns

Nothing

- Scope

All

setLotRefSequence(refSequence)

Description

Sets the sequence number corresponding to the lot.

Syntax

setLotRefSequence(refSequence)

- Parameters

Integer refSequence - The sequence number associated with the lot.

- Returns

Nothing

- Scope

All

setLotStatusFilter(lotStatusFilter)

Description

Set the custom lot status of results to return.

Syntax

setLotStatusFilter(lotStatusFilter)

- Parameters

String lotStatusFilter - The custom lot status value.

- Returns

Nothing

- Scope

All

setLotUUID(lotUUID)

Description

Sets the uuid corresponding to this material lot.

Syntax

setLotUUID(lotUUID)

- Parameters

String uuid - The unique identifier for this lot.

- Returns

Nothing

- Scope

All

setManualLotNo(manualLotNum)

Description

Sets the manually entered lot number.

Syntax

setManualLotNo(manualLotNum)

- Parameters

String manualLotNum - The lot number entered by the user.

- Returns

Nothing

- Scope

All

setMaterialRef(mesObjectLink)

Description

Sets the MES object link corresponding to this response material property.

Syntax

setMaterialRef(mesObjectLink)

- Parameters

MES Object Link mesObjectLink - The link corresponding to the material.

- Returns

Nothing

- Scope

All

setMaterialRefType(materialRefType)

Description

Sets the reference type of the material.

Syntax

setMaterialRefType(materialRefType)

- Parameters

S tring materialRefType - The type of the material reference.

- Returns

Nothing

- Scope

All

setMaterialRefUUID(materialRefUUID)

Description

Sets the uuid for the material reference.

Syntax

setMaterialRefUUID(materialRefUUID)

- Parameters

S tring materialRefUUID - The unique identifier for the material reference.

- Returns

Nothing

- Scope

All

setOptional(optional)

Description

Set the lot as an optional one.

Syntax

setOptional(optional)

- Parameters

boolean optional - Set to True if this lot should be optional and False otherwise.

- Returns

Nothing

- Scope

All

setProductionSelectable(productionSelectable)

Description

Sets the boolean value for the production selectable property. User can select the production if set to True.

Syntax

setProductionSelectable(productionSelectable)

- Parameters

boolean productionSelectable - Set to True if you need to enable this property and False otherwise.

- Returns

Nothing

- Scope

All

setQuantity(quantity)

Description

Sets the quantity set for the lot.

Syntax

setQuantity(quantity)

- Parameters

Double quantity - The actual quantity for this lot resource. This can be the current quantity at anytime during the life of a Response Segment, but will equal the final production quantity when this lot resource is finalized.

- Returns

Nothing

- Scope

All

setQuantitySource(quantitySource)

Description

This setting determines the source of the quantity for this response material resource.

Syntax

setQuantitySource(quantitySource)

- Parameters

String quantitySource - The name of the source of the quantity.

- Returns

Nothing

- Scope

All

setQuantitySourceLink(quantitySourceLink)

Description

Sets the name of the material resource to link to this segment. This is used when the Quantity Source setting is set to Link, Split or Combine.

Syntax

setQuantitySourceLink(quantitySourceLink)

- Parameters

String quantitySourceLink - The link to the quantity source.

- Returns

Nothing

- Scope

All

setQuantityUnits(quantityUnits)

Description

This property specifies the units for the quantity setting.

Syntax

setQuantityUnits(quantityUnits)

- Parameters

String quantityUnits - The units of quantity.

- Returns

Nothing

- Scope

All

setRate(rate)

Description

Sets the rate for the response material lot.

Syntax

setRate(rate)

- Parameters

Double rate - The rate to set for.

- Returns

Nothing

- Scope

All

setRatePeriod(ratePeriod)

Description

This is used to set the material rate period.

Options

Min - For setting the rate in minutes.

Hour - For setting the rate in hours.

Cycle - For setting the rate in cycles.

Syntax

setRatePeriod(ratePeriod)

- Parameters

String ratePeriod - The material rate period.

- Returns

Nothing

- Scope

All

setSegmentRefUUID(segmentRefUUID)

Description

Sets a reference uuid for the segment.

Syntax

setSegmentRefUUID(segmentRefUUID)

- Parameters

String segmentRefUUID - The uuid to set the segment for.

- Returns

Nothing

- Scope

All

setStatus(status)

Description

Sets the lot property status for the response material property.

Syntax

setStatus(status)

- Parameters

String status - The status to set the material property for.

- Returns

Nothing

- Scope

All

setUnits(units)

Description

Sets the units for the quantity setting.

Syntax

setUnits(units)

- Parameters

String units - The units of quantity.

- Returns

Nothing

- Scope

All

setUse(lotUse)

Description

Sets the material use types. Options are In, Out, Consumable, By-product.

Syntax

setUse(lotUse)

- Parameters

String lotUse - The lot use type for the response material.

- Returns

Nothing

- Scope

All

setZeroLotThresholdQty(lotNetQuantity)

Description

Sets the zero lot threshold quantity for the response material lot.

Syntax

setZeroLotThresholdQty(lotNetQuantity)

- Parameters

Double lotNetQuantity - The net quantity to be set to the material lot.

- Returns

Nothing

- Scope

All

MES Script Event

The MESScriptEvent object is passed when an MES object event is executed. Script for MES object events are defined in the Designer on the General Tab of the enterprise production item. See Events chapter. Optionally, the script defined for the event can execute the runDefaultHandler function on the event object to execute the built-in logic for the event.

From the MESScriptEvent object the full MES object can be loaded by calling the getMESObject function on the event.

Methods:

runDefaultHandler()

Description

Executes the built-in logic for the event. This allows pre or post logic to be executed around the built-in logic or completely replace the built-in logic.Note that user events will not have a default handler and no logic will be executed if this function is called.

Syntax

runDefaultHandler()

- Parameters

None

- Returns

Nothing

Properties:

getMESObject()

Description

Returns the MES object that the event is being executed for.

Syntax

getMESObject()

- Parameters

None

- Returns

AbstractMESObject - The MES object associated with the event.

getParameters()

Description

Return the MESObjectEventParameters object that contains any parameter values

Syntax

getParameters()

- Parameters

None

- Returns

MES Object Event Parameters - A new instance of a MESObjectEventParameters object.

setResult()

Description

Set the return result. When a MES events requires a result value, this helper function can be used in place of using the script event.getParameters().put('Result', value)

Syntax

setResult(value)

- Parameters

Value of the result to return

- Returns

Nothing

- Scope

All

MES User

Object Definition

The MES User object is used to get the user name and role list.

Methods

The following methods exist for the MES User object .

getRoleList()

Description

Gets the roles this MES user takes. It may be admin, operator, etc.

Syntax

getRoleList()

- Parameters

None

- Returns

List<String> role - The list of roles that are assigned to this MES user .

- Scope

All

getUserName()

Description

Gets the user name.

Syntax

getUserName()

- Parameters

None

- Returns

String name - The user name.

- Scope

All

MES Work Order

Base Object

The MES Work Order is derived from the MESAbstractObject and inherits all the exposed properties, methods and events for that object.

Scripting Functions

The following scripting functions return a MES Work Order ...

system.mes.workorder.createMESWorkOrder

system.mes.workorder.getMESWorkOrder

system.mes.workorder.getMESWorkOrders

Code Example

```
#Get a material link for the work order material
matLink = system.mes.getMESObjectLinkByName('MaterialDef', 'Soda')

#Create the work order
wo = system.mes.workorder.createMESWorkOrder('A34', matLink)

#Set some production related properties
wo.setWorkOrderQuantity(1000.0)

#Save the object
system.mes.saveMESObject(wo)
```

Methods

Beside the common MESAbstractObject methods, the following methods exist for the MES Work Order .

getActualQuantity()

Description

Gets the actual work order quantity.

Syntax

getActualQuantity()

- Parameters

None

- Returns

Double quantity - The work order quantity.

- Scope

All

getDueDate()

Description

Gets the due date for the work order.

Syntax

getDueDate()

- Parameters

None

- Returns

Date dueDate - The due date for the work order.

- Scope

All

getMaterialRef()

Description

Gets the material definition object corresponding to the work order.

Syntax

getMaterialRef()

- Parameters

None

- Returns

MESObjectLink mesMaterialDef - The material definition object corresponding to the work order.

- Scope

All

getMaterialRefName()

Description

Gets the name of the material reference object.

Syntax

getMaterialRefName()

- Parameters

None

- Returns

String materialRefName - Name of the material reference object.

- Scope

All

getMaterialRefType()

Description

Gets the type of the material reference object.

Syntax

getMaterialRefType()

- Parameters

None

- Returns

String materialRefType - Type of the material reference object.

- Scope

All

getMaterialRefUUID()

Description

Gets the uuid of the material reference object.

Syntax

getMaterialRefUUID()

- Parameters

None

- Returns

String materialRefUUID - The unique identifier that represent the material reference object.

- Scope

All

getRemainingQuantity()

Description

Gets the remaining quantity of the work order.

Syntax

getRemainingQuantity()

- Parameters

None

- Returns

Double qty - The remaining quantity for the work order.

- Scope

All

getScheduledQuantity()

Description

Gets the scheduled quantity of the work order.

Syntax

getScheduledQuantity()

- Parameters

None

- Returns

Double quantity - The scheduled quantity of the work order.

- Scope

All

getWorkOrderQuantity()

Description

Gets the quantity assigned for the work order.

Syntax

getWorkOrderQuantity()

- Parameters

None

- Returns

Double quantity - The quantity assigned for the work order.

- Scope

All

isClosed()

Description

Checks whether the work order is closed.

Syntax

isClosed()

- Parameters

None

- Returns

Boolean True if it is a closed work order and False otherwise.

- Scope

All

setClosed(closed)

Description

Sets the work order to closed.

Syntax

setClosed(closed)

- Parameters

Boolean closed - Set to True to set work order to close and False otherwise.

- Returns

Nothing

- Scope

All

setDueDate(dueDate)

Description

Sets the due date for the work order.

Syntax

setDueDate(dueDate)

- Parameters

Date dueDate - The due date to set for the work order.

- Returns

Nothing

- Scope

All

setMaterialRef(mesMaterialDef)

Description

Sets the material definition as a reference for the work order.

Syntax

setMaterialRef(mesMaterialDef)

- Parameters

MESMaterialDef mesMaterialDef - The material definition to set as the reference for the work order.

- Returns

Nothing

- Scope

All

setMaterialRef(materialLink)

Description

Sets the material reference for the work order.

Syntax

setMaterialRef(materialLink)

- Parameters

MESObjectLink materialLink - The material reference to set for the work order.

- Returns

Nothing

- Scope

All

setWorkOrderQuantity(quantity)

Description

Sets the quantity for the work order.

Syntax

setWorkOrderQuantity(quantity)

- Parameters

Double quantity - The quantity to set for the work order.

- Returns

Nothing

- Scope

All

Events

Besides the common MES object events, no other events exist for the MES Work Order .

Properties

Property values can be accessed and changed for an object by using the getPropertyValue() and setPropertyValue() method.

Example

```
obj = system.mes.MESObject('')     #Return a MES object
print obj.getPropertyValue('')
```

Besides the common core properties, the following properties exist for MES Work Order .

Setting Name	R/W	Description
MESMaterialRefProperty	Read /Write	The Material Definition assigned to the Work Order object.
MESMaterialRefUUIDProperty	Read /Write	The UUID of the Material Definition assigned to the Work Order object.
MESMaterialRefTypeProperty	Read /Write	The type of the Material Definition assigned to the Work Order object.
MESWorkOrderTargetQuantityProperty	Read /Write	The target quantity assigned to the Work Order object.
MESWorkOrderActualQuantityProperty	Read Only	The actual quantity assigned to the Work Order object.
MESWorkOrderScheduledQuantityProperty	Read Only	The scheduled quantity assigned to the Work Order object.
MESWorkOrderRemainingQuantityProperty	Read Only	The remaining quantity assigned to the Work Order object.
MESWorkOrderDueDateProperty	Read /Write	The due date assigned to the Work Order object.
MESWorkOrderClosedProperty	Read /Write	The boolean assigned to closed property of the Work Order object.

MES Work Order Filter

Object Description

MESWorkOrderFilter object is used to set filters for work order to narrow down the search results.

Scripting Functions

The following scripting function return a MES Work Order Filter ...

- system.mes.workorder.createMESWorkOrderFilter

Methods

Beside the common MESAbstractObject methods, the following methods exist for the MES Work Order Filter object .

getClosedBeginDate()

Description

Gets the closed begin date for the work order object.

Syntax

getClosedBeginDate()

- Parameters

None

- Returns

Date closedBeginDate - The closed begin date for the work order object.

- Scope

All

getClosedEndDate()

Description

Gets the closed end date for the MES work order object.

Syntax

getClosedEndDate()

- Parameters

None

- Returns

Date closedEndDate - The closed end date for the MES work order object.

- Scope

All

getEquipmentPathFilter()

Description

Gets the equipment path filter set for the MES work order object.

Syntax

getEquipmentPathFilter()

- Parameters

None

- Returns

String equipmentPathFilter - The equipment path filter associated with the work order object.

- Scope

All

getMaterialNameFilter()

Description

Gets the material name filter set for the MES work order object.

Syntax

getMaterialNameFilter()

- Parameters

None

- Returns

String nameFilter - The material name filter associated with the work order object.

- Scope

All

getReturnClosed()

Description

Checks for the boolean that is set for return closed property. The work order search results will include closed work orders if the return closed property is set to True.

Syntax

getReturnClosed()

- Parameters

None

- Returns

Boolaen returnClosed - The boolean that is set for the return closed property.

- Scope

All

getUpdateQuantities()

Description

Checks whether the quantities associated with the work order object are updated or not.

Syntax

getUpdateQuantities()

- Parameters

None

- Returns

Boolean updateQuantities - True if the quantities are updated and False otherwise.

- Scope

All

getWorkOrderNameFilter()

Description

Returns the name filter set for the MES Work Order object.

Syntax

getWorkOrderNameFilter()

- Parameters

None

- Returns

String workOrderNameFilter - The name filter that is set for the MES work order object.

- Scope

All

setClosedBeginDate(closedBeginDate)

Description

Sets the closed begin date for the work order object.

Syntax

setClosedBeginDate(closedBeginDate)

- Parameters

Date closedBeginDate - Date to be set as closed begin date for the work order object.

- Returns

Nothing

- Scope

All

setClosedEndDate(closedEndDate)

Description

Sets the closed end date for the work order object.

Syntax

setClosedEndDate(closedEndDate)

- Parameters

Date closedEndDate - Date to set as the closed end date.

- Returns

Nothing

- Scope

All

setEquipmentPathFilter(pathFilter)

Description

Sets the equipment path filter for the MES work order object.

Syntax

setEquipmentPathFilter(pathFilter)

- Parameters

String pathFilter - The equipment path filter to set for the work order object.

- Returns

Nothing

- Scope

All

setMaterialNameFilter(nameFilter)

Description

Sets the material name filter for the work order object.

Syntax

setMaterialNameFilter(nameFilter)

- Parameters

String nameFilter - The material name filter to set for the MES work order filter.

- Returns

Nothing

- Scope

All

setReturnClosed(returnClosed)

Description

Sets the return closed date for the work order object.

Syntax

setReturnClosed(returnClosed)

- Parameters

Boolean returnClosed - The boolean to set for the return closed property.

- Returns

Nothing

- Scope

All

setUpdateQuantities(updateQuantities)

Description

Sets the update quantity setting for the work order. Setting this property to True will update the quantities associated with this work order.

Syntax

setUpdateQuantities(updateQuantities)

- Parameters

Boolean updateQuantities - True to update the quantities for the work order and False otherwise.

- Returns

Nothing

- Scope

All

setWorkOrderNameFilter(searchPattern)

Description

Set the search pattern for the work order name.

Syntax

setWorkOrderNameFilter(searchPattern)

- Parameters

String searchPattern - The name of work order(s).

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
##Create a work order filter.  Get work orders based on the
filter.
##Print the list and the work order object in the list.
woName = "WO899"
woFilter = system.mes.workorder.createMESWorkOrderFilter()
woFilter.setWorkOrderNameFilter(woName)
results = system.mes.workorder.getMESWorkOrders(woFilter)
print results
```

```
for result in results:
    print result
```

Output

```
Size 1
WorkOrder (d709a509-932g-1382-b258-927d6ac4292c, WO899, 0
parents, 0 children, 0 custom properties, 0 complex
properties)
```

Modified MES Properties

The ModifiedMESProperties object holds the modified properties for an MES object. Whenever properties of a MES object are changed, only those changes may need to propagated to MES objects that were derived from the modified MES object. The system.mes.updateSegmentDependencies() script function takes a ModifiedMESProperties object as a parameter to updated any derived segment dependencies of the changed properties. By calling the getModifiedMESProperties() function on any MES object, a ModifiedMESProperties object containing modified properties since the last time it was saved will be returned.

Methods:

None

Properties:

None

Code Snippet

```
#This is an example of modifying a process segment and updating
all Process Segment and Operations Segment MES objects that
derived from it.

mesObject = system.mes.loadMESObject('Unload Vinegar', 'ProcessSeg
ment')
mesObject.setPropertyValue('Description', 'This is a new
description')
modProps = mesObject.getModifiedMESProperties()
system.mes.updateSegmentDependencies(mesObject, modProps)
```

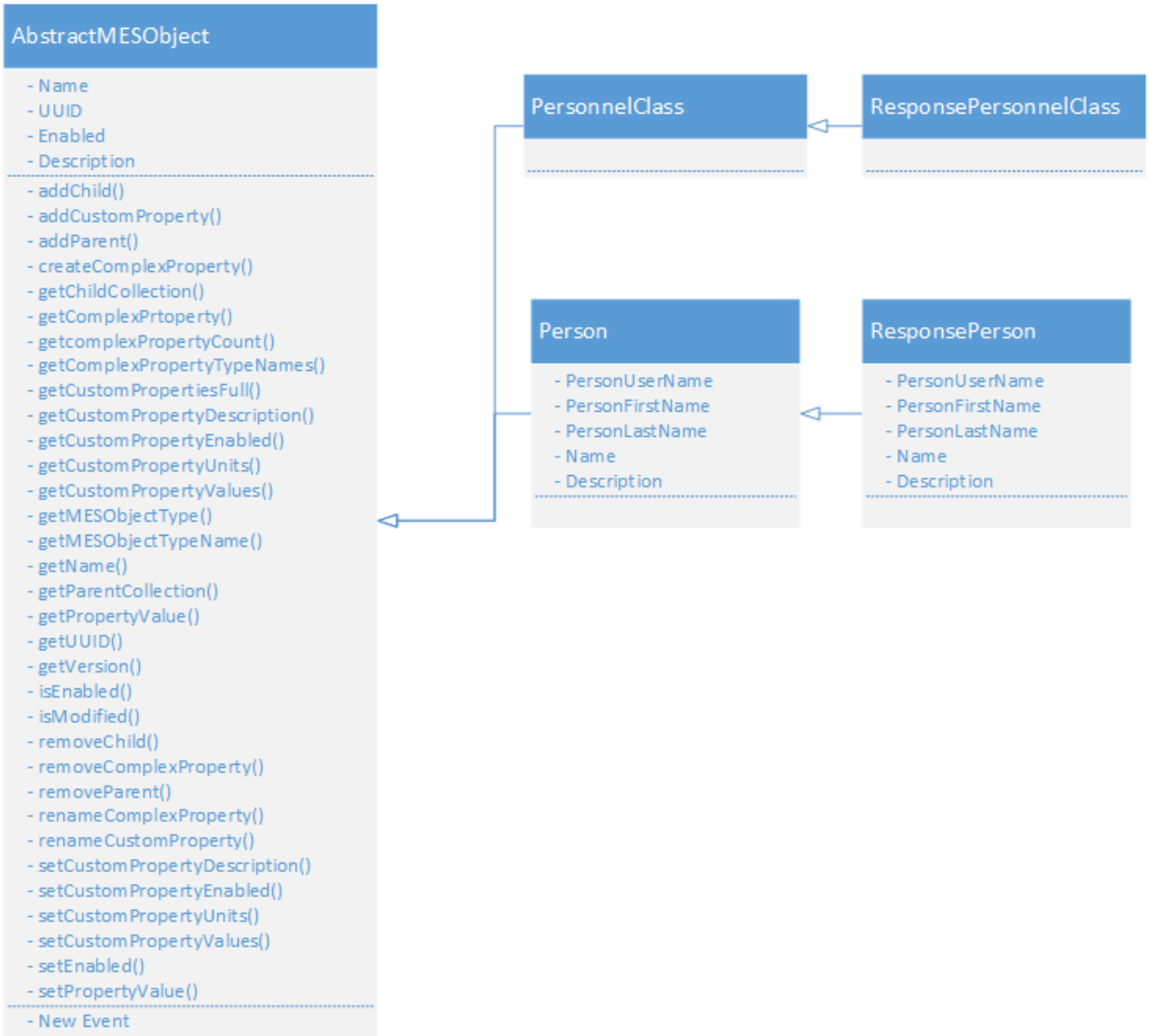
Personnel Objects

Any production or processing that is done involves people. In the Sepasoft MES system, this is optional. If trace information about the personnel involved in the production or processing is desired, then it is supported. The person can be automatically selected based on their Ignition login or it can be selected by another means.

There are two Object Types in the Personnel Objects Group, **Person** and **PersonnelClass**.

Person and **PersonnelClass** objects have a corresponding **ResponsePerson and ResponsePersonnelClass**, which is an internal versioning schema created to maintain historical production data whenever changes are made to the properties or settings of the Person and PersonnelClass objects.

All of these objects inherit the AbstractMESObject properties and methods. The **Person** and **ResponsePerson** further extend the parent object with the properties and methods found further down in this section.



Object Versions

Every time a Person or PersonnelClass Object is modified, i.e. adding custom properties, changing a setting etc., the version number of that equipment object will be updated in the background. When an operation is scheduled, it will check for a corresponding Response Object version. If one does not exist, it will automatically create a new Response object.

This versioning is not part of ISA-95, however, without it, analysis of historical data would lose the original configuration of equipment, personnel and materials.

For all intents and purposes, MES Person objects will be created and configured in the MES Management screen and through scripting. Response Objects are automatically created by Operations and will be used for any kind of traceability analysis.

Although these are called **Response** Objects, they are in fact **Version** objects of the **Person** Objects. They are not Response Segment objects as defined by ISA-95

PersonnelClass & ResponsePersonnelClass Objects

Defining production tasks for each specific person, is very tedious. A better method would be to organize the people into categories, or class using ISA-95 terms. An example will make this clearer with fewer words. Consider unloading vinegar at a unloading pump station. If there are ten operator who are qualified to unload vinegar, then creating a Vinegar Unload Operator class containing the ten qualified operators, only will require one unload vinegar task definition. Adding an eleventh operator is as simple as adding that person to the Vinegar Unload Operator class.

This object inherits the AbstractMESObject properties and methods, but does not extend them.

Person & ResponsePerson Objects

An object used to define people and cannot have children. The MES Person objects are automatically generated from the Ignition users that have first and last names defined. This prevent the default "admin" user from being created in the MES system and showing up in selection lists. When the Sepasoft MES modules first start, the MES Person objects are synchronized and then will be synchronized on a hourly basis thereafter. They can also synchronized on demand using a script function.

This object inherits the AbstractMESObject properties and methods. It also extends it with the following properties

Extended Properties

Setting Name	Description
Person User Name	The user name that the user logs in with.
Person First Name	The users first name that is configured in the Ignition user profile.
Person Last Name	The users last name that is configured in the Ignition user profile.
Name	This is the name of the MES object. This name is used when referencing the object. It must be a unique name.
Description	An optional settings to give more details for a MES Object.

Request Objects

Operations Schedule

Base Object

The Operations Schedule is derived from the MESAbstractObject and inherits all the exposed properties, methods and events for that object.

Object Description

The Operations Schedule is an object used for scheduling the operations.

Methods

The following methods exist for the Operations Schedule Object.

> **Info**
>
> These helper functions are available in version **1.8.3** or greater.

getScheduleCategory

Description

Returns the category of an operations schedule object.

Syntax

getScheduleCategory()

- Parameters

None

- Returns

String scheduleCategory - The category of the operations schedule object.

- Scope

All

getScheduleDurationSec

Description

Gets the duration of the schedule in seconds.

Syntax

getScheduleDurationSec()

- Parameters

None

- Returns

Integer scheduleDuration - The duration of schedule in seconds.

- Scope

All

getScheduleProductionCount

Description

Returns the production count for this schedule.

Syntax

getScheduleProductionCount()

- Parameters

None

- Returns

Double productionCount -The production count for this schedule.

- Scope

All

getSchedulePublishDate

Description

Returns date and time at which schedule was published. This setting is set when the operation is started, and it will be automatically set to the date and time of the Ignition server.

Syntax

getSchedulePublishDate()

- Parameters

None

- Returns

Date - The date and time the schedule was published.

- Scope

All

getScheduleType

Description

Gets the schedule type.

Syntax

getScheduleType()

- Parameters

None

- Returns

String type - The type of this schedule.

- Scope

All

setScheduleCategory

Description

Sets the category of an operations schedule object. Options for the toCategory parameter is either 'Hold' or 'Record Scheduled'.

Syntax

setScheduleCategory(scheduleCategory)

- Parameters

String scheduleCategory - The category of the operations schedule object.

- Returns

Nothing

- Scope

All

setScheduleDurationSec

Description

Sets the duration of the schedule in seconds.

Syntax

setScheduleDurationSec(scheduleDuration)

- Parameters

Integer scheduleDuration - The duration in seconds to set for.

- Returns

Nothing

- Scope

All

setScheduleProductionCount

Description

Sets the production count for this schedule.

Syntax

setScheduleProductionCount(productionCount)

- Parameters

Double productionCount -The production count to set for this schedule.

- Returns

Nothing

- Scope

All

setSchedulePublishDate

Description

Sets date and time at which schedule was published. This setting is set when the operation is started, and it will be automatically set to the date and time of the Ignition server.

Syntax

setSchedulePublishDate(schedulePublishDate)

- Parameters

Date schedulePublishDate - The date to set for.

- Returns

Nothing

- Scope

All

setScheduleType

Description

Sets the schedule type. This can be set to Held or Active.

Syntax

setScheduleType(scheduleType)

- Parameters

String scheduleType - The type to set for.

- Returns

Nothing

- Scope

All

Events

Besides the common MES object events, the following events exist for MES Work Order.

Setting Name	Description
BeginSchedule	The event is fired when an Operations Schedule has begun. If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()
EndSchedule	The event is fired when an Operations Schedule has ended. If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()
New	The event is fired when a new instance of an Operations Schedule object is created. If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()
UpdateProgress	The event is fired during the update progress of an Operations Schedule.

Setting Name	Description
	If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()

Properties

Besides the common core properties, the following properties exist for Operations Schedule Object.

Setting Name	R/W	Description
SchedulePublishDate	Read /Write	The date and time at which schedule was published. This setting is set when the operation is started, and it will be automatically set to the date and time of the Ignition server.
ScheduleType	Read /Write	This can be set to Held or Active.
ScheduleCategory	Read /Write	Changes the category of an operations schedule object. Options for the toCategory parameter is either 'Hold' or 'Record Scheduled'.
ScheduleProductionCount	Read /Write	The production for each schedule is tracked.
ScheduleDurationSec	Read /Write	This is the duration of the schedule in seconds.
RequestDependency	Read /Write	This property depends on operation request.
RequestRef	Read /Write	It refers to the operation that is requested for.
RequestRefUUID	Read /Write	

Setting Name	R/W	Description
		This setting is automatically set and should not be changed. It refers to the UUID of the Operations Definition or Operations Request, depending on how the operation was started. If the operation was scheduled prior to being run, then it will equal the UUID of the Operations Request it is based on, otherwise it will equal the UUID of the Operations Definition it is based on.
RequestRefType	Read /Write	It refers to the type of operation that is requested for.
WorkOrderRefUUID	Read /Write	A reference to an associated MES Work Order for the scheduled event(s).

Operations Request

Base Object

The Operations Request is derived from the MESAbstractObject and inherits all the exposed properties, methods and events for that object.

Scripting Functions

The following scripting functions return an Operations Request Object...

General Functions

There are many different types of MES objects in the Sepasoft MES system. All of them are inherited from the AbstractMESObject. Many of the scripting functions and properties refer to the commonAbstractMESObject objects. This page details the properties, functions and events that are common to all objects that are inherited from the AbstractMESObject.

Core Properties common to all MES Objects

Setting Name	Type	Description
name	read only	This is the name of the MES object. This name is used when referencing the object. It must be a unique name meaning that no other MES object of it's type can have the same name.
UUID		

Setting Name	Type	Description
	read only	This will contain the Universally Unique Identifier for each instance of a MES object.
enabled	write only	This property will be set to true when the MES object is active and usable. When MES objects are deleted they are still retained in the database and the Enabled setting is set to false. This is done to maintain past traceability information.
description	read-only	An optional settings to give more details for a MES Object.

Events

'New' Event

This event is run every time a new MES object is created. It can be used to add custom properties or to perform other tasks.

If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line:

Code Snippet

```
#Add custom property when a new instance of a MaterialDef object
is created.
obj = event.getMESObject()
obj.addCustomProperty('Width', 'Int4', 'Part Width', 'mm', True, F
alse)
event.runDefaultHandler()
```

Script Functions

The script functions listed below are available for all MES objects. They are used to simplify and reduce the number of lines of script for common tasks. An example is adding children, adding custom properties, changing property values, etc.

All of these script functions require an instance of a MES object. There are a number of methods to get an instance of an MES object and the code snippets below show just a couple of them.

Code Snippet

```
#Get the MES object for a given name and MES object type
obj = system.mes.loadMESObject('Balsamic Vinegar', 'MaterialDef')
```

Code Snippet

```
#If a link was returned from another script function, then this
will return the full MES object instance
obj = objLink.getMESObject()
```

Events

Besides the common MES object events, the following events exist for MES Work Order.

Setting Name	Description
BeforeAutoStart	The event is fired before the automatic start of a scheduled Operations Request. If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()
BeginSchedule	The event is fired before the requested operation is scheduled. If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()
EndSchedule	The event is fired after the requested operation has been scheduled. If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()
New	The event is fired when a new instance of an Operations Request object is created.

Setting Name	Description
	If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()
ScheduleDelay	The event is fired to give notification that an operation is delayed. If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()

Properties

Besides the common core properties, the following properties exist for Operations Request objects.

Setting Name	R/W	Description
Enable Update Event	Read /Write	When this setting is set to true, the UpdateProgress event for the Response Segments associated with this Operations Request will be executed at the interval set in the Update Event Interval. The UpdateProgress event is defined in the Ignition Designer in the MES Events section.
Update Event Interval	Read /Write	This setting defines the frequency (in ms) to execute the UpdateProgress event.
TrackProgressBy	Read /Write	Production can be tracked by two factors namely, time and material in each segment. The default is to track by time and the option to track by material will only show if in the associated process segment material has been defined and a rate has been specified.
IsExecuteReady	Read /Write	It is true only when there is a response segment to act upon. Operations Definition will not be ready for production unless all the segments are completely setup. And it is false if there is any error when validating the Operations Definition object.

Request Segment

Base Object

The Request Segment is derived from the MESAbstractObject and inherits all the exposed properties, methods and events for that object.

Methods

Beside the common MESAbstractObject methods, the following methods exist for the Request Segment.

getAvailableEquipmentOptions

> **Info**
>
> Available in **version** 1.8.3 or greater.

Description

Get a list of the available equipment options for the specified request segment.

Syntax

getAvailableEquipmentOptions()

- Parameters

None

- Returns

MESList<MESObjectLink> A list of MES Object Link objects holding the options that are appropriate for the segment. The list is returned as a MES List object that is a collection holding MES object links that represent the options.

- Scope

All

getAvailableLotEquipmentOptions(baseName)

> **Info**

Available in **version** 1.8.3 or greater.

Description

Get a list of the available lot equipment options for the specified request segment.

Syntax

getAvailableLotEquipmentOptions(baseName)

- Parameters

String baseName - The base name for the complex property associated with this segment. Complex properties that use extended naming have an associated base name and a number following it. This is used with Lot Reference Property that the Request Segment uses. Each time a lot reference is created it is named the base name with an extension added to it.

- Returns

MESList<MESObjectLink> A list of MES Object Link objects holding the options that are appropriate for the segment. The list is returned as a MES List object that is a collection holding MES object links that represent the options.

- Scope

All

getAvailableMaterialOptions(baseName)

Info

Available in **version** 1.8.3 or greater.

Description

Get a list of the available material options for the specified request segment.

Syntax

getAvailableMaterialOptions(baseName)

- Parameters

String baseName - The base name for the complex property associated with this segment. Complex properties that use extended naming have an associated base name and a number following it. This is used with Lot Reference Property that the Request Segment uses. Each time a lot reference is created it is named the base name with an extension added to it.

- Returns

MESList<MESObjectLink> A list of MES Object Link objects holding the options that are appropriate for the segment. The list is returned as a MES List object that is a collection holding MES object links that represent the options.

- Scope

All

getAvailablePersonnelOptions(baseName)

Info

Available in **version** 1.8.3 or greater.

Description

Get a list of the available personnel options for the specified request segment.

Syntax

getAvailablePersonnelOptions(baseName)

- Parameters

String baseName - The base name for the complex property associated with this segment. Complex properties that use extended naming have an associated base name and a number following it. This is used with Lot Reference Property that the Request Segment uses. Each time a lot reference is created it is named the base name with an extension added to it.

- Returns

MESList<MESObjectLink> A list of MES Object Link objects holding the options that are appropriate for the segment. The list is returned as a MES List object that is a collection holding MES object links that represent the options.

- Scope

All

getAvailableSupplementalEquipmentOptions(baseName)

Info

Available in **version** 1.8.3 or greater.

Description

Get a list of the available supplemental equipment options for the specified request segment.

Syntax

getAvailableSupplementalEquipmentOptions(baseName)

- Parameters

String baseName - The base name for the complex property associated with this segment. Complex properties that use extended naming have an associated base name and a number following it. This is used with Lot Reference Property that the Request Segment uses. Each time a lot reference is created it is named the base name with an extension added to it.

- Returns

MESList<MESObjectLink> A list of MES Object Link objects holding the options that are appropriate for the segment. The list is returned as a MES List object that is a collection holding MES object links that represent the options.

- Scope

All

getEquipment

Description

Return the equipment MES object associated with the segment. This will be the same equipment that is associated with the operation that the segment is running under.

Syntax

getEquipment()

- Parameters

None

- Returns

The AbstractMESObject representing the equipment.

- Scope

All

getEquipmentLink

Description

Return the link to the equipment MES object associated with the segment. This will be the same equipment that is associated with the operation that the segment is running under.

Syntax

getEquipmentLink()

- Parameters

None

- Returns

The MES Object Link representing the equipment.

- Scope

All

getEquipmentProperty

Description

Return the complex property of the equipment MES object associated with the segment. This will be the same equipment that is associated with the operation that the segment is running under.

Syntax

getEquipmentProperty()

- Parameters

None

- Returns

The MESEquipmentProperty representing the equipment.

- Scope

All

setEquipmentLink(equipmentLink)

Info

Available in **version** 1.8.3 or greater.

Description

Set the equipment that the segment is associated with by the link to the equipment.

Syntax

setEquipmentLink(equipmentLink)

- Parameters

MES Object LinkMES Object Linkequipment Link - A MES Object Link object containing details of the equipment to associate with the segment.

- Returns

Nothing

- Scope

All

setMaterial(baseName, materialName, equipmentPath)

Info

Available in **version** 1.8.3 or greater.

Description

This script function is used to set the material resources for a segment by material and location. There are different versions depending on if a lot is being referenced, a new lot is being created, etc. It can be used before or after the begin script function is called on the segment. If they are used for an active segment, then they will update the material resources information. This is common when changing to a different lot of material or other material related information during a production run.

Syntax

setMaterial(baseName, materialName, equipmentPath)

- Parameters

String baseName - The base name for the complex property associated with this segment. Complex properties that use extended naming have an associated base name and a number following it. This is used with Lot Reference Property that the Request Segment uses. Each time a lot reference is created it is named the base name with an extension added to it.

String materialName - The material name that must match an existing MESMaterialDef object.

String equipmentPath - The equipment path that must match the path to equipment defined in the production model.

- Returns

Nothing

- Scope

All

setMaterial(baseName, materialName, equipmentPath, quantity)

Info

Available in **version** 1.8.3 or greater.

Description

This script function is used to set the material resources for a segment. There are different versions depending on if a lot is being referenced, a new lot is being created, etc. It can be used before or after the begin script function is called on the segment. If they are used for an active segment, then they will update the material resources information. This is common when changing to a different lot of material or other material related information during a production run.

Syntax

setMaterial(baseName, materialName, equipmentPath, quantity)

- Parameters

String baseName - The base name for the complex property associated with this segment. Complex properties that use extended naming have an associated base name and a number following it. This is used with Lot Reference Property that the Request Segment uses. Each time a lot reference is created it is named the base name with an extension added to it.

String materialName - The material name that must match an existing MESMaterialDef object.

String equipmentPath - The equipment path that must match the path to equipment defined in the production model.

Double quantity - The quantity of material.

- Returns

Nothing

- Scope

All

setMaterial(baseName, materialName, equipmentPath, quantity, customProperties)

Info

Available in **version** 1.8.3 or greater.

Description

This script function is used to set the material resources for a segment. There are different versions depending on if a lot is being referenced, a new lot is being created, etc. It can be used before or after the begin script function is called on the segment. If they are used for an active segment, then they will update the material resources information. This is common when changing to a different lot of material or other material related information during a production run.

Syntax

setMaterial(baseName, materialName, equipmentPath, quantity, customProperties)

- Parameters

String baseName - The base name for the complex property associated with this segment. Complex properties that use extended naming have an associated base name and a number following it. This is used with Lot Reference Property that the Request Segment uses. Each time a lot reference is created it is named the base name with an extension added to it.

String materialName - The material name that must match an existing MESMaterialDef object.

String equipmentPath - The equipment path that must match the path to equipment defined in the production model.

Double quantity - The quantity of material.

PyDictionary customProperties - A PyDictionary containing either name value pairs or complete custom property definitions. See Custom Properties for more information.

- Returns

Nothing

- Scope

All

setMaterial(baseName, materialName, equipmentPath, manualLotNumber, quantity)

Info

Available in **version** 1.8.3 or greater.

Description

This version of the setMaterial script function is used for setting the material name and location equipment to use the material from, or place the material at for a material reference belonging to a segment. For cases when a material lot does already exist, this script function is used and allows for naming the new lot and updating the quantity at the same time. Instead of automatically generating a new lot number, the system with use the manual lot number provided in the manualLotNumber parameter.

The material name must match the name of an existing MESMaterialDef object which corresponds to the ISA-95 Material Definition object. The equipment path must match a valid line, line cell, line cell group or storage unit defined in the production model. See the custom properties for more information.

Syntax

setMaterial(baseName, materialName, equipmentPath, manualLotNumber, quantity)

- Parameters

String baseName - The base name for the complex property associated with this segment. Complex properties that use extended naming have an associated base name and a number following it. This is used with Lot Reference Property that the Request Segment uses. Each time a lot reference is created it is named the base name with an extension added to it.

String materialName - The material name that must match an existing MESMaterialDef object.

String equipmentPath - The equipment path that must match the path to equipment defined in the production model.

String manualLotNumber - The lot number to name to the new MESMaterialLot object

Double quantity - The quantity of material.

- Returns

Nothing

- Scope

All

setMaterial(baseName, materialName, equipmentPath, manualLotNumber, quantity, customProperties)

Info

Available in **version** 1.8.3 or greater.

Description

This version of the setMaterial script function is used for setting the material name and location equipment to use the material from, or place the material at for a material reference belonging to a segment. For cases when a material lot does already exist, this script function is used and allows for naming the new lot and updating the quantity at the same time. Instead of automatically generating a new lot number, the system with use the manual lot number provided in the manualLotNumber parameter.

The material name must match the name of an existing MESMaterialDef object which corresponds to the ISA-95 Material Definition object. The equipment path must match a valid line, line cell, line cell group or storage unit defined in the production model. See the custom properties for more information.

Syntax

setMaterial(baseName, materialName, equipmentPath, manualLotNumber, quantity, customProperties)

- Parameters

String baseName - The base name for the complex property associated with this segment. Complex properties that use extended naming have an associated base name and a number following it. This is used with Lot Reference Property that the Request Segment uses. Each time a lot reference is created it is named the base name with an extension added to it.

String materialName - The material name that must match an existing MESMaterialDef object.

String equipmentPath - The equipment path that must match the path to equipment defined in the production model.

String manualLotNumber - The lot number to name to the new MESMaterialLot object

Double quantity - The quantity of material.

PyDictionary customProperties - A PyDictionary containing either name value pairs or complete custom property definitions. See Custom Properties for more information.

- Returns

Nothing

- Scope

All

setPersonnel(baseName, personName)

> **Info**
>
> Available in **version** 1.8.3 or greater.

Description

This script function is used to set the personnel resources for a segment. It can be used before or after begin script function is called on the segment. If they are used for an active segment, then they will update the personnel resources information. This is common when changing to different personnel during a production run.

Syntax

setPersonnel(baseName, personName)

- Parameters

String baseName - The base name for the complex property associated with this segment. Complex properties that use extended naming have an associated base name and a number following it. This is used with Lot Reference Property that the Request Segment uses. Each time a lot reference is created it is named the base name with an extension added to it.

String personName - The name of the person. The name of the person is derived from the last and first name from the Ignition user list.

- Returns

Nothing

- Scope

All

setPersonnel(baseName, personName, customProperties)

> **Info**
>
> Available in **version** 1.8.3 or greater.

Description

This script function is used to set the personnel resources for a segment. It can be used before or after begin script function is called on the segment. If they are used for an active segment, then they will update the personnel resources information. This is common when changing to different personnel during a production run.

Syntax

setPersonnel(baseName, personName, customProperties)

- Parameters

String baseName - The base name for the complex property associated with this segment. Complex properties that use extended naming have an associated base name and a number following it. This is used with Lot Reference Property that the Request Segment uses. Each time a lot reference is created it is named the base name with an extension added to it.

String personName - The name of the person. The name of the person is derived from the last and first name from the Ignition user list.

PyDictionary customProperties - A PyDictionary containing either name value pairs or complete custom property definitions. See Custom Properties for more information.

- Returns

Nothing

- Scope

All

setSupplementalEquipment(baseName, equipmentName)

Info

Available in **version** 1.8.3 or greater.

Description

This script function is used to set the supplemental equipment resources for a segment.

Syntax

setSupplementalEquipment(baseName, equipmentName)

- Parameters

String baseName - The base name for the complex property associated with this segment. Complex properties that use extended naming have an associated base name and a number following it. This is used with Lot Reference Property that the Request Segment uses. Each time a lot reference is created it is named the base name with an extension added to it.

String equipmentName - The name of the supplemental equipment.

- Returns

Nothing

- Scope

All

setSupplementalEquipment(baseName, equipmentName, customProperties)

Info

Available in **version** 1.8.3 or greater.

Description

Set the supplemental equipment that the segment is associated with.

Syntax

setSupplementalEquipment(baseName, equipmentName, customProperties)

- Parameters

String baseName - The base name for the complex property associated with this segment. Complex properties that use extended naming have an associated base name and a number following it. This is used with Lot Reference Property that the Request Segment uses. Each time a lot reference is created it is named the base name with an extension added to it.

String equipmentName - The name of the supplemental equipment.

PyDictionary customProperties - A PyDictionary containing either name value pairs or complete custom property definitions. See Custom Properties for more information.

- Returns

Nothing

- Scope

All

Events

Besides the common MES object events, the following events exist for Request Segment.

Setting Name	Description
New	The Schedule Request Segment event is fired when a Request Segment is begin scheduled. This script must determine the begin date time and end date time of the request segment. If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()
Schedule	The Schedule Request Segment event is fired when a Request Segment is begin scheduled. This script must determine the begin date time and end date time of the request segment. If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()

Properties

Besides the common core properties, the following properties exist for Request Segment objects.

Setting Name	R/W	Description
Operation Request Ref UUID	Read /Write	This setting is automatically set and should not be changed. It refers to the UUID of the Operations Request.
Begin Date Time	Read /Write	The date and time the operation started. This setting is set when the operation is started, and it will automatically be set to the date and time of the Ignition server.
End Date Time	Read /Write	The date and time the operation ended. This setting is set when the operation is ended, and it will automatically be set to the date and time of the Ignition server.
Operation Segment Ref Type	Read /Write	This setting is automatically set and should not be changed. It will either be set to Operations Segment or Request Segment depending on how the operation was started. If the operation was scheduled prior to being run, then it will equal Request Segment, otherwise it will equal Operations Segment.
Operation Segment Ref UUID	Read /Write	This setting is automatically set and should not be changed. It refers to the UUID of the Operations Segment or Request Segment, depending on how the operation was started. If the operation was scheduled prior to being run, then it will equal the UUID of the Request Segment it is based on, otherwise it will equal the UUID of the Operations Segment it is based on.
Segment Execute Enabled	Read /Write	When this setting is set to true, the Segments associated with this Operations Request will be executed.
End Operation When Complete	Read /Write	This setting will end the operation automatically, after completion.

Response Objects

Operations Performance

Base Object

The Operations Performance is derived from the MESAbstractObject and inherits all the exposed properties, methods and events for that object.

Methods

Beside the common MESAbstractObject methods, the following methods exist for the Operations Performance objects.

getActiveSegmentName

Description

Return the name of the segment which is currently active.

Syntax

getActiveSegmentName()

- Parameters

None

- Returns

An MES Object Link representing the active segment.

- Scope

All

getEquipmentLink

Description

Return the link to the equipment MES object associated with the segment. This will be the same equipment that is associated with the operation that the segment is running under.

Syntax

getEquipmentLink()

- Parameters

None

- Returns

The MES Object Link representing the equipment.

- Scope

All

Events

Besides the common MES object events, no other events exist for the Operations Performance object.

Properties

Besides the common core properties, the following properties exist for Operations Performance object.

Setting Name	R/W	Description
Schedule Reference UUID	Read /Write	The UUID of the schedule associated with this operation.
Schedule Publish Date Time	Read /Write	The date and time the schedule was published. This setting is automatically set to the date and time of the Ignition server.
ScheduleType	Read /Write	This can be set to Held or Active.
ScheduleCategory	Read /Write	Changes the category of a operations schedule object. Options for the toCategory parameter is either 'Hold' or 'Record Scheduled'.
ScheduleProductionCount		The production for each schedule is tracked.

Setting Name	R/W	Description
	Read /Write	
ScheduleDurationSec	Read /Write	This is the duration of the schedule in seconds.

Operations Response

Base Object

The Operations Response is derived from the MESAbstractObject and inherits all the exposed properties, methods and events for that object.

Extended Script Functions

This object inherits the properties and methods of MESAbstractObject and extends the available methods as shown below.

- abort
- begin
- createSegment
- end
- getActiveSegment
- getActiveSegmentName
- getActiveSegmentNames
- getAvailableSegmentNames
- getEquipmentLink
- getWorkOrderLink
- setEquipmentLink
- setWorkOrderLink

All of these script functions require an instance of a MESOperationsResponse object. The following code snippets are a couple of examples of how to get an instance of a MESOperationsResponse.

```
Code Snippet
```

```
#Get the currently running operation for specified equipment
eqPath = '[global]\Dressings Inc\California\Raw Materials\Unload
Station 1'
oper = system.mes.getCurrentOperation(eqPath)
```

Code Snippet

```
#Create new operation for specified equipment
eqPath = '[global]\Dressings Inc\California\Raw Materials\Unload
Station 1'
oper = system.mes.createOperation('Receive Steel')
```

Object Description

The Operations Response is an object that is created behind the scenes anytime a Operations Definition is selected to begin. It hold the actual production or traceability information. It can also be created from script and used to begin segments from script, but it cannot be created from the components provided with the Track and Trace Module. For this reason, there is no component that can be used to change the properties listed for the Operations Response object.

Events

Besides the common MES object events, the following events exist for MES Work Order.

Setting Name	Description
New	The event is fired when a new instance of an Operations Response object is created. If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()

Properties

Besides the common core properties, the following properties exist for Operations Response.

Setting Name	R/W	Description
	Read /Write	

Setting Name	R/W	Description
Enable Update Event		When this setting is set to true, the UpdateProgress event for the Response Segments associated with this Operations Definition will be executed at the interval set in the Update Event Interval. The UpdateProgress event is defined in the Ignition Designer in the MES Events section.
Update Event Interval	Read /Write	This setting defined the frequency (in mS) to execute the UpdateProgress event.
Begin Date Time	Read /Write	The date and time the operation started. This setting is set when the operation is started, and it will automatically be set to the date and time of the Ignition server.
End Date Time	Read /Write	The date and time the operation ended. This setting is set when the operation is ended, and it will automatically be set to the date and time of the Ignition server.
Operation Reference Type	Read /Write	This setting is automatically set and should not be changed. It will either be set to Operations Definition or Operations Request depending on how the operation was started. If the operation was scheduled prior to being run, then it will equal Operations Request, otherwise it will equal Operations Definition.
Operation Reference UUID	Read /Write	This setting is automatically set and should not be changed. It refers to the UUID of the Operations Definition or Operations Request, depending on how the operation was started. If the operation was scheduled prior to being run, then it will equal the UUID of the Operations Request it is based on, otherwise it will equal the UUID of the Operations Definition it is based on.

abort

Note

This helper function should **NOT** be used in normal operations because the tracking data will not be validated or in many cases accurately recorded in the database.

Description

Abort the operation. This will do an abrupt abort of all segments currently running under the operation before aborting the operation.

Syntax

abort()

- Parameters

None

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Get the MES object link of Unload Station 1
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'

#Get the current operation running at the equipment path
oper = system.mes.getCurrentOperation(eqPath)

#Abort the operation
oper.abort()
```

begin

Description

Begin the operation which will allow segments to begin. Only one operation can be running at an equipment item at a time, but multiple segments can run under an operation.

Syntax

begin()

- Parameters

None

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'

#Get a new operation to be run at the specified equipment path
oper = system.mes.createOperation('Receive Steel', eqPath)

#Begin the operation
oper.begin()
```

Overview

These script functions are used to create a new segment.

createSegment(segmentName)

Description

Create a new segment to run under the operation. After calling this script function, the segment will not be running until required resources and custom properties are assigned begin is called. Operations can contain one or more segments and the name of the segment to create is passed in the segmentName parameter.

Syntax

createSegment(segmentName)

- Parameters

String segmentName - The name of the segment to create.

- Returns

A new MESResponseSegment object that can be used to assign resources and custom properties.

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Create and begin a new operation at the specified equipment
path
oper = system.mes.createOperation('Receive Steel', eqPath)
oper.begin()
#Create new segment
seg = oper.createSegment('Inspect Steel')
#Assign material resources....
#Assign personnel resources....
#Assign supplemental resources....
#Assign custom properties....
#Begin segment
seg.begin()
```

createSegment(segmentName, autoAssignOptions)

Description

Create a new segment to run under the operation. After calling this script function, the segment will not be running until required resources and custom properties are assigned begin is called. Operations can contain one or more segments and the name of the segment to create is passed in the segmentName parameter.

Syntax

createSegment(segmentName, autoAssignOptions)

- Parameters

String segmentName - The name of the segment to create.

Boolean autoAssignOptions - If true, automatically assign material, lot and person options. Otherwise, they have to be set prior to beginning the segment.

- Returns

A new MESResponseSegment object that can be used to assign resources and custom properties.

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Create and begin a new operation at the specified equipment
path
oper = system.mes.createOperation('Receive Steel', eqPath)
oper.begin()
#Create new segment
seg = oper.createSegment('Inspect Steel', True)
#Assign material resources....
#Assign personnel resources....
#Assign supplemental resources....
#Assign custom properties....
#Begin segment
```

```
seg.begin()
```

end

Description

End the operation. Before an operation can be ended all segments must be ended first.

Syntax

end()

- Parameters

None

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'

#Get the currently running operation running at the specified
equipment path
oper = system.mes.getCurrentOperation(eqPath)

#End the operation
oper.end()
```

getActiveSegment

Description

Get the active segment MES object from a operation by the name of the segment. Because more than one segment can be running under an operation, the segmentName parameter specifies which one to return.

Syntax

getActiveSegment(segmentName)

- Parameters

String segmentName - The name of the segment.

- Returns

The MESResponseSegment object that can be updated to reflect lot, material, personnel, supplemental equipment or custom properties changes.

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'

#Get the current operation being run at the equipment
oper = system.mes.getCurrentOperation(eqPath)

#Get the Inspect Steel segment running under the operation
seg = oper.getActiveSegment('Inspect Steel')

#End the segment
seg.end()
```

getActiveSegmentName

Description

Return the name of currently active segment for the operation. If there are no active segments or more than one active segment, then an exception will be thrown.

Syntax

getActiveSegmentName()

- Parameters

None

- Returns

The name of the currently active segment.

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'

#Get the current operation being run at the equipment
oper = system.mes.getCurrentOperation(eqPath)

#Get the active segment running under the operation
segName = oper.getActiveSegmentName()
seg = oper.getActiveSegment(segName)

#End the segment
seg.end()
```

getActiveSegmentNames

Description

Return names of all active segments for an operation. Operations can be running multiple segments at any time and this script function is used to get the names of them.

Syntax

getActiveSegmentNames()

- Parameters

None

- Returns

An array of the names of all active segments for the operation.

- Scope

All

Code Examples

Code Snippet

```
#Get the operation currently being run at Unload Station 1
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
oper = system.mes.getCurrentOperation(eqPath)
operName = oper.getName()

#Get the names of active segments for the operation
if oper == None:
    print 'No active segments found for "%s"' % eqPath
else:
    try:
        segNames = oper.getActiveSegmentNames()
        for name in segNames:
            print name

    except:
        print 'There are no segments available for "%s "' %
operName
```

Output

```
Unload Steel

Inspect Steel
```

getAvailableSegmentNames

Description

Return a list of all available segments that can be run for the operation. Both currently running and not running segments will be returned.

Syntax

getAvailableSegmentNames()

- Parameters

None

- Returns

An array of the names of all available segments for the operation.

Code Examples

Code Snippet

```
#Get the operation currently being run at Unload Station 1
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
oper = system.mes.getCurrentOperation(eqPath)

#Get the names of available segments for the operation
segNames = oper.getAvailableSegmentNames()
for name in segNames:
    print name
```

Output

```
Unload Steel
Inspect Steel
Store in Inventory
```

getEquipmentLink

Description

Return a link to the equipment MES object associated with the operation.

Syntax

getEquipmentLink()

- Parameters

None

- Returns

MES Object Link object containing details of the equipment that is associated with the operation.

- Scope

All

Code Examples

Code Snippet

```
#Get the operation currently being run at Unload Station 1
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
oper = system.mes.getCurrentOperation(eqPath)

#Get the MES object link of where the operation is running
eqLink = oper.getEquipmentLink()

#Print the UUID and the type of equipment
print eqLink.getMESObjectUUID()
print eqLink.getMESObjectType()
```

```
#Get the full MES object for the equipment
eqObj = eqLink.getMESObject()
print eqObj.getName()
```

Output

```
508ebebc-8f19-4521-b018-a3e177028981
Response Line
Unload Station 1
```

getWorkOrderLink

Description

Return a link to the work order MES object associated with the operation.

Syntax

getWorkOrderLink()

- Parameters

None

- Returns

MESObjectLink object containing details of the work order that is associated with the operation.

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Enterprise\San Marcos\MP Rotator\MP Rotator
1'
oper = system.mes.getCurrentOperation(eqPath)
#Get work order link
print oper.getWorkOrderLink()
```

Output

```
(type: Work Order,  uuid: 2e906f1e-1736-4e02-8586-6b7e4e7e17fc)
```

setEquipmentLink

Description

Set the equipment that the operation is associated with by the link to the equipment.

Notice

Using this method is not the preferred method of assigning the equipment of where an operation is to run. Instead, use the system.mes.createOperation script function that will set the equipment based on the equipment path.

Syntax

setEquipmentLink(equipmentLink)

- Parameters

A MES Object Link object containing details of the equipment to associate with the operation.

- Returns

Nothing

- Scope

All

Code Examples

Non Preferred Method

```
#Get the MES object link of Unload Station 1
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
eqLink = system.mes.getMESObjectLinkByEquipmentPath(eqPath)

#Create a new operation from a operation definition object
operDef = system.mes.loadMESObject('Receive Steel', 'Operations
Definition')
oper = system.mes.createOperation(operDef)

#Set the equipment where the new operation is to run.
oper.setEquipmentLink(eqLink)
```

Preferred Method

```
#Get the MES object link of Unload Station 1
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'

#Create a new operation using the equipment path
oper = system.mes.createOperation('Receive Steel', eqPath)
```

setWorkOrderLink

Description

Sets a link to the work order MES object associated with the operation.

Syntax

setWorkOrderLink(woLink)

- Parameters

MESObjectLink woLink - The object link containing details of the work order that is associated with the operation.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet

eqPath = '[global]\Enterprise\San Marcos\MP Rotator\MP Rotator
1'
#Get current operation
oper = system.mes.getCurrentOperation(eqPath)

#Get the work order link
woLink=system.mes.getMESObjectLinkByName('WorkOrder', 'New
Work Order')

#Set the work order link
oper.setWorkOrderLink(woLink)
```

Response Segment

Base Object

The Response Segment is derived from the MESAbstractObject and inherits all the exposed properties, methods and events for that object.

Object Description

The Response Segment is an object that is created behind the scenes anytime Operations Segment is selected to begin. It holds the actual production or traceability information. It can also be created from the script and used to begin segments from script and not from the components provided with the Track and Trace Module. For this reason, there is not a component that can be used to change the properties listed for the Response Segment objects.

Extended Script Functions

This object inherits the AbstractMESObject properties and methods and also extend it with the same properties and methods as shown.

abort

Description

Abort the segment. This will do an abrupt abort of the segments. The operation the segment is running under and all other segments running under the operation will remain active.

Notice

This helper function should **NOT** be used in normal operations because the tracking data will not be validated or in many cases accurately recorded in the database. For normal operations use the end segment, which will be accurately recorded in the database.

Syntax

abort()

- Parameters

None

- Returns

Nothing

- Scope

All

addSublot(materialPropertyName, sublotName)

Description

These script functions are used to add sublots one at a time to active segments and the different versions provide various methods to do so. Sublots are represented by MESMaterialSublot objects which corresponds to the ISA-95 Material Sublot objects. MESMaterialSublot objects must be children of a MESMaterialLot object, event though the

lot information may not be needed. Usually, when production details are maintained for serialized items moving through production as groups, sublots are used. In cases where each item moves independently through production, then just material lots can be used for each serialized item.

Notice

In order for material sublots to be added, the Enable Sublots setting of the segment material property must be set to true.

addSublot(materialPropertyName, sublotName)

Description

This version of the addSublot script function is used to create a single new material sub lot. When the new is created, the CreateSerialNumber event of the MaterialSublot object will be executed and a serial number will automatically be assigned. This serial number, which is the name of the MaterialSublot object, can be changed prior updating the segment.

Syntax

addSublot(materialPropertyName, sublotName)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

String sublotName - The name of the sublot which is usually the serial number of the item.

- Returns

MESMaterialSublot - A new MESMaterialSublot object.

- Scope

All

Code Examples

Code Snippet

```
#Create a new segment
seg = system.mes.createSegment('Load Assembly Tray', '[global]
\Dressings Inc\California\Assembly\PS Assembly', False)
#Assign the material resources
seg.setMaterial('Housing', 'Housing', 'Assembly Tray 8')
#Begin the segment
seg.begin()
#Because the reference to the segment is different than the
one at the Ignition gateway after begin was executed, refresh
it.
seg = system.mes.getActiveSegment('[global]\Dressings
Inc\California\Assembly\PS Assembly', 'Load Assembly Tray')
#Create new sublots
seg.addSublot('Housing', 'SN 1234')
seg.addSublot('Housing', 'SN 2345')
seg.update()
```

addSublot(materialPropertyName, sublotName, customProperties)

Description

This version of the addSublot script function functions the same as the addSublot (materialPropertyName, sublotName) script function above with the added support to assign new custom properties to the new material sublot at the same time.

Syntax

addSublot(materialPropertyName, sublotName, customProperties)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

String sublotName - The name of the sublot which is usually the serial number of the item.

PyDictionary customProperties - A PyDictionary containing either name value pairs or complete custom property definitions. See Custom Properties for more information.

- Returns

MESMaterialSublot - A new MESMaterialSublot object.

- Scope

All

Code Examples

Code Snippet

```
#Create a new segment
seg = system.mes.createSegment('Load Assembly Tray', '[global]
\Dressings Inc\California\Assembly\PS Assembly', False)
#Assign the material resources
seg.setMaterial('Housing', 'Housing', 'Assembly Tray 8')
#Begin the segment
seg.begin()
#Because the reference to the segment is different than the
one at the Ignition gateway after begin was executed, refresh
it.
seg = system.mes.getActiveSegment('[global]\Dressings
Inc\California\Assembly\PS Assembly', 'Load Assembly Tray')
#Create new sublot
#Add custom properties
#Custom property definition format: {custom_property_name:
[ignition_data_type, value, description, units,
production_visible, required}
cp = {'Width' : ['Int4', 1020, 'Width of housing', 'mm', True,
True], 'Height' : ['Int4', 800, 'Height of housing', 'mm', True
, True]}
seg.addSublot('Housing', 'SN 1234', cp)
#Create second sublot with different custom property values
cp = {'Width' : ['Int4', 1025, 'Width of housing', 'mm', True,
True], 'Height' : ['Int4', 790, 'Height of housing', 'mm', True
, True]}
seg.addSublot('Housing', 'SN 2345', cp)
seg.update()
```

addSublots(materialPropertyName, countToAdd)

Description

These script functions are used to add multiple sublots to active segments and the different versions provide various methods to do so. Sublots are represented by MESMaterialSublot objects which corresponds to the ISA-95 Material Sublot objects. MESMaterialSublot objects must be children of a MESMaterialLot object, event though the lot information may not be needed. Usually, when production details are maintained for serialized items moving through production as groups, sublots are used. In cases where each item moves independently through production, then just material lots can be used for each serialized item.

Notice

In order for material sublots to be added, the Enable Sublots setting of the segment material property must be set to true.

Description

This version of the addSublots script function is used to create a specified quantity of new material sublots. For each sublot object created, the CreateSerialNumber event of the MaterialSublot object will be executed and a serial number will automatically be assigned. This serial number, which is the name of the MaterialSublot object, can be changed prior updating the segment.

Syntax

addSublots(materialPropertyName, countToAdd)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

Integer countToAdd - The number of sub lots to add.

- Returns

List<MESMaterialSublot> - A list of new MESMaterialSublot objects. The size of the list will match the countToAdd parameter.

- Scope

All

Code Examples

Code Snippet

```
#Create a new segment
seg = system.mes.createSegment('Load Assembly Tray', '[global]
\Dressings Inc\California\Assembly\PS Assembly', False)
#Assign the material resources
seg.setMaterial('Housing', 'Housing', 'Assembly Tray 8')
#Begin the segment
seg.begin()
#Because the reference to the segment is different than the
one at the Ignition gateway after begin was executed, refresh
it.
seg = system.mes.getActiveSegment('[global]\Dressings
Inc\California\Assembly\PS Assembly', 'Load Assembly Tray')
#Create new sublots
sublotList = seg.addSublots('Housing', 5)
for index in range(sublotList.size()):
    #Print the automatically generated serial number
    print sublotList.get(index).getName()
seg.update()
```

begin

Description

Begin the segment. Multiple segments can be running under one operation, but only one operation can be running at an equipment item at a time.

Syntax

begin()

- Parameters

None

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet

eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Create and begin a new operation at the specified equipment
path
oper = system.mes.createOperation('Receive Steel', eqPath)
oper.begin()
#Create new segment
seg = oper.createSegment('Inspect Steel')
#Assign material resources....
#Assign personnel resources....
#Assign supplemental resources....
#Assign custom properties....
#Begin segment
seg.begin()
```

changeLot

Description

Changes the lot to a new lot property.

Syntax

changeLot(lotPropertyName)

- Parameters

String lotPropertyName - Name of the lot property to be changed.

- Returns

MESResponseMaterialProperty newLotProp - The new lot property.

- Scope

All

Code Examples

```
Code Snippet
```

```
Output
```

end

Description

End an active segment. Multiple segments can be running under one operation, but only one operation can be running at an equipment item at a time.

Syntax

end()

- Parameters

None

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Get the current operation being run at the equipment
oper = system.mes.getCurrentOperation(eqPath)
#Get the active segment running under the operation
segName = oper.getActiveSegmentName()
seg = oper.getActiveSegment(segName)
#End the segment
seg.end()
```

execute

Description

Execute a segment. For situations where a production task or event is happening with no definite begin and end times, the execute script function will begin and end a segment with one script function. After the execute script function is called on a segment, the segment will be left in the inactive state and no further updates to it can be done.

Syntax

execute()

- Parameters

None

- Returns

Nothing

- Scope

All

getAvailableMaterialLots

Description

Gets the material lots which are currently available.

Syntax

getAvailableMaterialLots(materialPropertyName, equipmentPathFilter)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

String equipmentPathFilter - Equipment path to filter results.

- Returns

MESList<MESObjectLink> - A list of MESObjectLink containing lot number or lot sequence number of available material lots.

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Nuts Unlimited\Folsom\Receiving\Nut
Unloading'
#Create an operation
oper = system.mes.createOperation('Unload Nuts', eqPath)
oper.begin()
#Create a segment
seg = oper.createSegment('Unload Nuts')
seg.setMaterial('Received Nuts', 'Bulk Almonds', '[global]
\Nuts Unlimited\Folsom\Receiving\Nut Storage Silos\Almond Silo'
)
#Gets the available material lots
availLot=seg.getAvailableMaterialLots('Received Nuts', '[global
]\Nuts Unlimited\Folsom\Receiving\Nut Storage Silos\Almond
Silo')
#Returns the lot number of the available material lot
for ndx in range(availLot.size()):
        availLotNumber = availLot.get(ndx)
print availLotNumber
```

Output

```
OperationsResponse (3b7d2e8d-ca90-4e8f-9682-c364fd2ec991,
Unload Nuts, 0 parents, 0 children, 0 custom properties, 1
complex properties)
A 2234
```

getEquipmentLink

Description

Return the link to the equipment MES object associated with the segment. This will be the same equipment that is associated with the operation that the segment is running under.

Syntax

getEquipmentLink()

- Parameters

None

- Returns

A MES Object Link object containing details of the equipment that is associated with the segment.

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Get the current operation being run at the equipment
oper = system.mes.getCurrentOperation(eqPath)
```

```
#Get the active segment running under the operation
segName = oper.getActiveSegmentName()
seg = oper.getActiveSegment(segName)
#Get the MES object link of where the operation is running
eqLink = seg.getEquipmentLink()
#Print the UUID and the type of equipment
print eqLink.getMESObjectUUID()
print eqLink.getMESObjectType()
#Get the full MES object for the equipment
eqObj = eqLink.getMESObject()
print eqObj.getName()
```

Output

```
508ebebc-8f19-4521-b018-a3e177028981
Response Line
Unload Station 1
```

getLot

Description

Returns the response material object corresponding to the given name parameter.

Syntax

getLot(lotPropertyName)

- Parameters

String lotPropertyName - The property associated with the lot to return for.

- Returns

MESResponseMaterialProperty - The name of the response material object corresponding to the given lotPropertyName parameter.

- Scope

All

Code Examples

Code Snippet

getMaterialLot

Description

Return the material lot MES object associated the specified material property of a segment. Material lots is represented by MESMaterialLot objects and correspond to ISA-95 Material Lot objects. MESMaterialLot objects are automatically created by the Sepasoft MES system when segments are began, updated, executed or ended. In cases where additional information is attached to the material lot, this script function provides an easy method of getting the actual MESMaterialLot object.

Syntax

getMaterialLot(materialPropertyName)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

- Returns

A MESMaterialLot object that is associated with the specified material property for segment.

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
```

```
#Get the current operation being run at the equipment
oper = system.mes.getCurrentOperation(eqPath)
#Get the active segment running under the operation
segName = oper.getActiveSegmentName()
seg = oper.getActiveSegment(segName)
#Get the material lot object associated with the Housing
material property of the segment
lot = seg.getMaterialLot('Housing')
#Add custom properties to the material lot object
#Custom property definition format: {custom_property_name:
[ignition_data_type, value, description, units,
production_visible, required}
cp = {'Width' : ['Int4', 1003, 'Width of housing', 'mm', True,
True], 'Height' : ['Int4', 788, 'Height of housing', 'mm', True
, True]}
lot.setCustomPropertyValues(cp)
#Remember to save the material lot object
system.mes.saveMESObject(lot)
```

getPerson

Description

Return the person MES object associated with the specified personnel property of a segment. People is represented by MESPerson objects and correspond to ISA-95 Person objects. The MESResponsePerson object isolates changes made to a MESPerson object and production history. In cases where additional information is attached the the person, this script function provides an easy method of getting the actual MESResponsePerson object.

Syntax

getPerson(personnelPropertyName)

- Parameters

String personnelPropertyName - The name of the personnel property item as defined for the segment.

- Returns

A MESResponsePerson object that is associated with the specified personnel property for segment.

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Get the current operation being run at the equipment
oper = system.mes.getCurrentOperation(eqPath)
#Get the active segment running under the operation
segName = oper.getActiveSegmentName()
seg = oper.getActiveSegment(segName)
#Get the material lot object associated with the Housing
material property of the segment
person = seg.getPerson('Inspector')
#Add custom properties to the person object
#Custom property definition format: {custom_property_name:
[ignition_data_type, value, description, units,
production_visible, required}
cp = {'Backup' : True}
person.setCustomPropertyValues(cp)
#Remember to save the material lot object
system.mes.saveMESObject(person)
```

getSegmentEquipmentLink

Description

Get the link to segment equipment.

Syntax

getSegmentEquipmentLink()

- Parameters

None

- Returns

The MES Object Link object .

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Create and begin a new operation at the specified equipment
path
oper = system.mes.createOperation('Receive Steel', eqPath)
oper.begin()
#Create new segment
seg = oper.createSegment('Inspect Steel')
#Assign material resources....
#Assign personnel resources....
#Assign supplemental resources....
#Assign custom properties....
#Execute segment
seg.execute()
```

getSubLot

Description

Get an existing sublot that is associated with a segment. Sublots are represented by MESMaterialSublot objects and correspond to the ISA-95 Material Sublot objects.

Syntax

getSublot(materialPropertyName, sublotName)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

String sublotName - The name of an existing sublot to return.

- Returns

MESMaterialSublot - The MESMaterialSublot object.

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Get the current operation being run at the equipment
oper = system.mes.getCurrentOperation(eqPath)
#Get the active segment running under the operation
segName = oper.getActiveSegmentName()
seg = oper.getActiveSegment(segName)
#Get the sublot with serial number SN1823
sublot = seg.getSublot('Housing', 'SN1823')
#Do something with the sublot
sublot.setPropertyValue('Width', '1002')
#Don't forget to save the changes to the sublot
system.mes.saveMESObject(sublot)
```

getSupplementalEquipment

Description

Return the supplemental equipment MES object associated with the specified supplemental equipment property of a segment. Supplemental equipment is represented by MESEquipment objects and correspond to ISA-95 Equipment objects. The MESResponseEquipment object isolates changes made to a MESEquipment object and production history. In cases where additional information is attached to the supplemental, this script function provides an easy method of getting the actual MESResponseEquipment object.

Syntax

getSupplementalEquipment(supplementalEquipmentPropertyName)

- Parameters

String supplementalEquipmentPropertyName - The name of the supplemental equipment property item as defined for the segment.

- Returns

MESResponseEquipment - A MESResponseEquipment object that is associated with the specified supplemental equipment property for segment.

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Get the current operation being run at the equipment
oper = system.mes.getCurrentOperation(eqPath)
#Get the active segment running under the operation
segName = oper.getActiveSegmentName()
seg = oper.getActiveSegment(segName)
#Get the material lot object associated with the Housing
material property of the segment
supEq = seg.getSupplementalEquipment('Die')
#Add custom properties to the supplemental equipment object
#Custom property definition format: {custom_property_name:
[ignition_data_type, value, description, units,
production_visible, required}
cp = {'Wear' : 12}
supEq.setCustomPropertyValues(cp)
#Remember to save the material lot object
system.mes.saveMESObject(supEq)
```

removeSublot

Description

Remove an existing sublot that is associated with a segment. Sublots are represented by MESMaterialSublot objects and correspond to the ISA-95 Material Sublot objects.

Syntax

removeSublot(materialPropertyName, sublotName)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

String s ublotName - The name of an existing sublot to remove.

- Returns

The MESMaterialSublot object that was removed.

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Get the current operation being run at the equipment
oper = system.mes.getCurrentOperation(eqPath)
#Get the active segment running under the operation
segName = oper.getActiveSegmentName()
seg = oper.getActiveSegment(segName)
#Remove the sublot with serial number SN1823
seg.removeSublot('Housing', 'SN1823')
```

renameSublot

Description

Rename an existing sublot that is associated with a segment. Sublots are represented by MESMaterialSublot objects and correspond to the ISA-95 Material Sublot objects.

Syntax

renameSublot(materialPropertyName, existingSublotName, newSublotName)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

String existingS ublotName - The existing name of the sublot, which is usually the serial number of the item, to rename.

String newSublotName - The new name for the sublot.

- Returns

The MESMaterialSublot object that was renamed.

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Get the current operation being run at the equipment
oper = system.mes.getCurrentOperation(eqPath)
#Get the active segment running under the operation
segName = oper.getActiveSegmentName()
seg = oper.getActiveSegment(segName)
#Change the sublot serial number from SN9823 to SN1823
seg.renameSublot('Housing', 'SN9823', 'SN1823')
```

setMaterial

Description

These script functions are used to set the material resources for a segment. There are different versions depending on if a lot is being referenced, a new lot is being created, etc. These script functions can be used before or after the begin script function is called on the segment. If they are used for an active segment, then they will update the material resources information. This is common when changing to a different lot of material or other material related information during a production run.

Existing Lot

setMaterial(materialPropertyName, quantity)

Description

This version of the setMaterial script function is used for setting just the quantity for a material reference belonging to a segment. If the lot or material have already been assigned, then this provides a method to update the quantity.

Syntax

setMaterial(materialPropertyName, quantity)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

Double quantity - The quantity of material.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet
```

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Create and begin a new operation at the specified equipment
path
oper = system.mes.createOperation('Receive Steel', eqPath)
oper.begin()
#Create new segment
seg = oper.createSegment('Inspect Steel')
#Assign material resources....
seg.setMaterial('In Steel Type', 1045)
#Begin segment
seg.begin()
```

setMaterial(materialPropertyName, lotNumber)

Description

This version of the setMaterial script function is used for setting the lot number for a material reference belonging to a segment. The material lot must already exist for the lot number at the equipment location as defined in the material reference. If the lot has already been assigned, then this provides a method to change the lot.

Syntax

setMaterial(materialPropertyName, lotNumber)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

String lotNumber - The lot number that must match the name of an existing MESMaterialLot object.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Create and begin a new operation at the specified equipment
path
oper = system.mes.createOperation('Receive Steel', eqPath)
oper.begin()
#Create new segment
seg = oper.createSegment('Inspect Steel')
#Assign material resources....
seg.setMaterial('In Steel Type', 'SL1285')
#Begin segment
seg.begin()
```

setMaterial(materialPropertyName, lotNumber, quantity)

Description

This version of the setMaterial script function is used for setting the lot number and quantity for a material reference belonging to a segment. The material lot must already exist for the lot number at the equipment location as defined in the material reference. If the lot has already been assigned, then this provides a method to change the lot and update the quantity at the same time.

Syntax

setMaterial(materialPropertyName, lotNumber, quantity)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

String lotNumber - The lot number that must match the name of an existing MESMaterialLot object.

Double quantity - The quantity of material.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet

eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Create and begin a new operation at the specified equipment
path
oper = system.mes.createOperation('Receive Steel', eqPath)
oper.begin()
#Create new segment
seg = oper.createSegment('Inspect Steel')
#Assign material resources....
seg.setMaterial('In Steel Type', 'SL1285', 1000.0)
#Begin segment
seg.begin()
```

setMaterial(materialPropertyName, lotNumber, lotSequenceNumber, quantity)

Description

This version of the setMaterial script function is used for setting the lot number with a lot sequence number and quantity for a material reference belonging to a segment. The material lot and lot sequence number combination must already exist for the lot number at the equipment location as defined in the material reference. If the lot has already been assigned, then this provides a method to change the lot and update the quantity at the same time.

Material lots are represented by MESMaterialLot objects which corresponds to the ISA-95 Material Lot objects. Because more that one MESMaterialLot object can existing for a given lot number, each one is assigned a unique lot sequence number making it unique. Normally this is not required provided that only one lot number (MESMaterialLot object) exists at an equipment locations. If more than one MESMaterialLot object with the same lot

number exists at an equipment location, then the lot sequence number can be used to specify which one to use. If more than one do exist and the lot sequence number is not specified, then the one used is determined by the Lot Handling Mode the equipment location is configured for.

Syntax

setMaterial(materialPropertyName, lotNumber, lotSequenceNumber, quantity)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

String lotNumber - The lot number that must match the name of an existing MESMaterialLot object.

Integer lotSequenceNumber - The lot sequence number of the desired existing MESMaterialLot object. If it is less than zero, then the lot holding the maximum value is returned and if the same lot number is used for multiple segments, each lot with the same lot number will be assigned a different lot sequence number. It could be obtained by using getLotSequence() script function. For more details see MES Lot Quantity Summary Item.

Double quantity - The quantity of material.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Create and begin a new operation at the specified equipment
path
oper = system.mes.createOperation('Receive Steel', eqPath)
oper.begin()
#Create new segment
seg = oper.createSegment('Inspect Steel')
```

```
#Get the lot sequence number.
equipLotSummary = system.mes.getLotInventoryByEquipment('[globa
l]\My Enterprise\California\Storage\Vinegar Tanks\Vinegar Tank
1')
for lotSummary in equipLotSummary:
    lotSeq = lotSummary.getLotSequence()
#Assign material resources....
seg.setMaterial('In Steel Type', 'SL1285', lotSeq, 1000.0)
#Begin segment
seg.begin()
```

setMaterial(materialPropertyName, lotNumber, lotSequenceNumber, quantity, customProperties)

Description

This version of the setMaterial script function functions the same as the setMaterial (materialPropertyName, lotNumber, lotSequenceNumber, quantity) script function above with the added support to assign custom properties for the material resource reference at the same time.

Syntax

setMaterial(materialPropertyName, lotNumber, lotSequenceNumber, quantity, customProperties)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

String lotNumber - The lot number that must match the name of an existing MESMaterialLot object.

Integer lotSequenceNumber - The lot sequence number of the desired existing MESMaterialLot object. If it is less than zero, then the lot holding the maximum value is returned and if the same lot number is used for multiple segments, each lot with the same lot number will be assigned a different lot sequence number. It could be obtained by using getLotSequence() script function. For more details see MES Lot Quantity Summary Item .

Double quantity - The quantity of material.

PyDictionary customProperties - A PyDictionary containing either name value pairs or complete custom property definitions. See Custom Properties for more information.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
 eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Create and begin a new operation at the specified equipment
path
oper = system.mes.createOperation('Receive Steel', eqPath)
oper.begin()
#Create new segment
seg = oper.createSegment('Inspect Steel')
#Get the lot sequence number
equipLotSummary = system.mes.getLotInventoryByEquipment('[globa
l]\My Enterprise\California\Storage\Vinegar Tanks\Vinegar Tank
1')
for lotSummary in equipLotSummary:
    lotSeq = lotSummary.getLotSequence()
#Assign material resources....
#Assign values to existing custom properties
cp = {'Thickness' : 5.5}
seg.setMaterial('In Steel Type', 'SL1285', lotSeq, 1000.0, cp)
#Begin segment
seg.begin()
```

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Create and begin a new operation at the specified equipment
path
oper = system.mes.createOperation('Receive Steel', eqPath)
oper.begin()
#Create new segment
seg = oper.createSegment('Inspect Steel')
```

```
#Get the lot sequence number
equipLotSummary = system.mes.getLotInventoryByEquipment('[globa
l]\My Enterprise\California\Storage\Vinegar Tanks\Vinegar Tank
1')
for lotSummary in equipLotSummary:
    lotSeq = lotSummary.getLotSequence()
#Assign material resources....
#Add custom properties
#Custom property definition format: {custom_property_name:
[ignition_data_type, value, description, units,
production_visible, required}
cp = {'Thickness' : ['Float8', 5.5, 'Thickness of steel', '1
/1000th', True, True]}
seg.setMaterial('In Steel Type', 'SL1285', lotSeq, 1000.0, cp)
#Begin segment
seg.begin()
```

Non-existent Lot

setMaterial(materialPropertyName, materialName, equipmentPath, quantity)

Description

This version of the setMaterial script function is used for setting the material name and location equipment to use the material from, or place the material at for a material reference belonging to a segment. For cases when a material lot does already exist, this script function is used and allows for updating the quantity at the same time. With this version of the setMaterial script function, the lot number is automatically generated in the CreateLotNumber event of the MESMaterialLot object.

The material name must match the name of an existing MESMaterialDef object which corresponds to the ISA-95 Material Definition object. The equipment path must match a valid line, line cell, line cell group or storage unit defined in the production model. See Custom Properties for more information.

Syntax

setMaterial(materialPropertyName, materialName, equipmentPath, quantity)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

String materialName - The material name that must match an existing MESMaterialDef object.

String equipmentPath - The equipment path that must match the path to equipment defined in the production model.

Double quantity - The quantity of material.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Create and begin a new operation at the specified equipment
path
oper = system.mes.createOperation('Receive Steel', eqPath)
oper.begin()
#Create new segment
seg = oper.createSegment('Unload Steel')
#Assign material resources....
seg.setMaterial('In Steel Type', 'Pre-inspected Hardened Steel'
, 'Inspection Staging', 1000.0)
#Begin segment
seg.execute()
```

setMaterial(materialPropertyName, materialName, equipmentPath, quantity, customProperties)

Description

This version of the setMaterial script function functions the same as the setMaterial (materialPropertyName, materialName, equipmentPath, quantity) script function above with the added support to assign custom properties for the material resource reference at the same time.

Syntax

setMaterial(materialPropertyName, materialName, equipmentPath, quantity, customProperties)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

String materialName - The material name that must match an existing MESMaterialDef object.

String equipmentPath - The equipment path that must match the path to equipment defined in the production model.

Double quantity - The quantity of material.

PyDictionary c ustomProperties - A PyDictionary containing either name value pairs or complete custom property definitions. See the custom properties for more information.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Create and begin a new operation at the specified equipment
path
oper = system.mes.createOperation('Receive Steel', eqPath)
oper.begin()
```

```
#Create new segment
seg = oper.createSegment('Unload Steel')
#Assign material resources....
#Assign values to existing custom properties
cp = {'Thickness' : 5.5}
seg.setMaterial('In Steel Type', 'Pre-inspected Hardened Steel'
, 'Inspection Staging', 1000.0, cp)
#Begin segment
seg.execute()
```

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Create and begin a new operation at the specified equipment
path
oper = system.mes.createOperation('Receive Steel', eqPath)
oper.begin()
#Create new segment
seg = oper.createSegment('Unload Steel')
#Assign material resources....
#Assign values to existing custom properties
#Add custom properties
#Custom property definition format: {custom_property_name:
[ignition_data_type, value, description, units,
production_visible, required}
cp = {'Thickness' : ['Float8', 5.5, 'Thickness of steel', '1
/1000th', True, True]}
seg.setMaterial('In Steel Type', 'Pre-inspected Hardened Steel'
, 'Inspection Staging', 1000.0, cp)
#Begin segment
seg.execute()
```

setMaterial(materialPropertyName, materialName, equipmentPath, quantity, customProperties)

Description

This version of the setMaterial script function functions the same as the setMaterial (materialPropertyName, materialName, equipmentPath, quantity) script function above with the added support to assign custom properties for the material resource reference at the same time.

Syntax

setMaterial(materialPropertyName, materialName, equipmentPath, quantity, customProperties)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

String materialName - The material name that must match an existing MESMaterialDef object.

String equipmentPath - The equipment path that must match the path to equipment defined in the production model.

Double quantity - The quantity of material.

PyDictionary customProperties - A PyDictionary containing either name value pairs or complete custom property definitions. See the custom properties for more information.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Create and begin a new operation at the specified equipment
path
oper = system.mes.createOperation('Receive Steel', eqPath)
oper.begin()
#Create new segment
seg = oper.createSegment('Unload Steel')
#Assign material resources....
#Assign values to existing custom properties
cp = {'Thickness' : 5.5}
seg.setMaterial('In Steel Type', 'Pre-inspected Hardened Steel'
, 'Inspection Staging', 1000.0, cp)
#Begin segment
seg.execute()
```

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Create and begin a new operation at the specified equipment
path
oper = system.mes.createOperation('Receive Steel', eqPath)
oper.begin()
#Create new segment
seg = oper.createSegment('Unload Steel')
#Assign material resources....
#Assign values to existing custom properties
#Add custom properties
#Custom property definition format: {custom_property_name:
[ignition_data_type, value, description, units,
production_visible, required}
cp = {'Thickness' : ['Float8', 5.5, 'Thickness of steel', '1
/1000th', True, True]}
seg.setMaterial('In Steel Type', 'Pre-inspected Hardened Steel'
, 'Inspection Staging', 1000.0, cp)
#Begin segment
seg.execute()
```

setMaterial(materialPropertyName, materialName, equipmentPath, manualLotNumber, quantity)

Description

This version of the setMaterial script function is used for setting the material name and location equipment to use the material from, or place the material at for a material reference belonging to a segment. For cases when a material lot does already exist, this script function is used and allows for naming the new lot and updating the quantity at the same time. Instead of automatically generating a new lot number, the system with use the manual lot number provided in the manualLotNumber parameter.

The material name must match the name of an existing MESMaterialDef object which corresponds to the ISA-95 Material Definition object. The equipment path must match a valid line, line cell, line cell group or storage unit defined in the production model. See the custom properties for more information.

Syntax

setMaterial(materialPropertyName, materialName, equipmentPath, manualLotNumber, quantity)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

String materialName - The material name that must match an existing MESMaterialDef object.

String equipmentPath - The equipment path that must match the path to equipment defined in the production model.

String manualLotNumber - The lot number to name to the new MESMaterialLot object

Double quantity - The quantity of material.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
 eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Create and begin a new operation at the specified equipment
path
oper = system.mes.createOperation('Receive Steel', eqPath)
oper.begin()
#Create new segment
seg = oper.createSegment('Unload Steel')
#Assign material resources....
seg.setMaterial('In Steel Type', 'Pre-inspected Hardened Steel'
, 'Inspection Staging', 'SL8923', 1000.0)
#Begin segment
seg.execute()
```

setMaterial(materialPropertyName, materialName, equipmentPath, manualLotNumber, quantity, customProperties)

Description

This version of the setMaterial script function functions the same as the setMaterial (materialPropertyName, materialName, equipmentPath, manualLotNumber, quantity) script function above with the added support to assign custom properties for the material resource reference at the same time.

Syntax

setMaterial(materialPropertyName, materialName, equipmentPath, manualLotNumber, quantity, customProperties)

- Parameters

String materialPropertyName - The name of the material property item as defined for the segment.

String materialName - The material name that must match an existing MESMaterialDef object.

String equipmentPath - The equipment path that must match the path to equipment defined in the production model.

String manualLotNumber - The lot number to name to the new MESMaterialLot object

Double quantity - The quantity of material.

PyDictionary customProperties - A PyDictionary containing either name value pairs or complete custom property definitions . See the custom properties for more information.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet
```

```
 eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Create and begin a new operation at the specified equipment
path
oper = system.mes.createOperation('Receive Steel', eqPath)
oper.begin()
#Create new segment
seg = oper.createSegment('Unload Steel')
#Assign material resources....
#Assign values to existing custom properties
cp = {'Thickness' : 5.5}
seg.setMaterial('In Steel Type', 'Pre-inspected Hardened Steel'
, 'Inspection Staging', 'SL8923', 1000.0, cp)
#Begin segment
seg.execute()
```

Code Snippet

```
 eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Create and begin a new operation at the specified equipment
path
oper = system.mes.createOperation('Receive Steel', eqPath)
oper.begin()
#Create new segment
seg = oper.createSegment('Unload Steel')
#Assign material resources....
#Add custom properties
#Custom property definition format: {custom_property_name:
[ignition_data_type, value, description, units,
production_visible, required}
cp = {'Thickness' : ['Float8', 5.5, 'Thickness of steel', '1
/1000th', True, True]}
seg.setMaterial('In Steel Type', 'Pre-inspected Hardened Steel'
, 'Inspection Staging', 'SL8923', 1000.0, cp)
#Begin segment
seg.execute()
```

setMaterialByPassChecks(baseName, lotNumber)

Overview

Description

Sets the lot for the segment's material reference bypassing inventory checks.

Click here to read the Knowledge base article for bypassing inventory checks.

Info

The Lot Availability Status must be set to "Available" which means this script function is for an active segment object.

Method Options

Syntax

setMaterialByPassChecks(baseName, lotNumber)

- Parameters

String baseName - The base name for the complex property.

String lotNumber - The lot number that must match the name of an existing MESMaterialLot object.

- Returns

Nothing

- Scope

All

Syntax

setMaterialBypassChecks(baseName, lotSequenceNumber, lotNumber)

- Parameters

String baseName - The base name for the complex property.

String lotSequenceNumber - The lot sequence number corresponding to the material lot object. If it is less than zero, then the lot holding the maximum value is returned and if the same lot number is used for multiple segments, each lot with the same lot number will be assigned a different lot sequence number.

String lotNumber - The lot number that must match the name of an existing MESMaterialLot object.

- Returns

Nothing

- Scope

All

Syntax

setMaterialBypassChecks(baseName, lotNumber, quantity)

- Parameters

String baseName - The base name for the complex property.

String lotNumber - The lot number that must match the name of an existing MESMaterialLot object.

Double quantity - The quantity of material.

- Returns

Nothing

- Scope

All

Syntax

setMaterialBypassChecks(baseName, lotNumber, lotSequenceNumber, quantity)

- Parameters

String baseName - The base name for the complex property.

String lotNumber - The lot number that must match the name of an existing MESMaterialLot object.

String lotSequenceNumber - The lot sequence number corresponding to the material lot object. If it is less than zero, then the lot holding the maximum value is returned and if the same lot number is used for multiple segments, each lot with the same lot number will be assigned a different lot sequence number.

Double quantity - The quantity of material.

- Returns

Nothing

- Scope

All

Syntax

setMaterialBypassChecks(baseName, lotNumber, lotSequenceNumber, quantity, customProperties)

- Parameters

String baseName - The base name for the complex property.

String lotNumber - The lot number that must match the name of an existing MESMaterialLot object.

String lotSequenceNumber - The lot sequence number corresponding to the material lot object. If it is less than zero, then the lot holding the maximum value is returned and if the same lot number is used for multiple segments, each lot with the same lot number will be assigned a different lot sequence number.

Double quantity - The quantity of material.

PyDictionary customProperties - A PyDictionary containing either name value pairs or complete custom property definitions. See Custom Properties for more information.

- Returns

Nothing

- Scope

All

setPersonnel

Description

These script functions are used to set the personnel resources for a segment. There are different versions detailed below. These script functions can be used before or after begin script function is called on the segment. If they are used for an active segment, then they will update the personnel resources information. This is common when changing to different personnel during a production run.

Syntax

setPersonnel(personnelPropertyName, personName)

- Parameters

String personnelPropertyName - The name of the personnel property item as defined for the segment.

String personName - The name of the person. The name of the person is derived from the last and first name from the Ignition user list.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Create and begin a new operation at the specified equipment
path
oper = system.mes.createOperation('Receive Steel', eqPath)
oper.begin()
#Create new segment
seg = oper.createSegment('Inspect Steel')
#Assign peronnel resources....
seg.setPersonnel('Inspector', 'Smith, Sue')
#Begin segment
seg.begin()
```

setSupplementalEquipment(supplementalEquipmentPropertyName, equipmentName, customProperties)

Description

This version of the setSupplementalEquipment script function functions the same as the setSupplementalEquipment(supplementalEquipmentPropertyName, equipmentName) script function above with the added support to assign custom properties for the supplemental equipment resource reference at the same time.

Syntax

setSupplementalEquipment(supplementalEquipmentPropertyName, equipmentName, customProperties)

- Parameters

String supplementalEquipmentPropertyName - The name of the supplemental equipment property item as defined for the segment.

String equipmentName - The name of the supplemental equipment.

PyDictionary customProperties - A PyDictionary containing either name value pairs or complete custom property definitions. See Custom Properties for more information.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\ABC Inc\California\Pressing\Press 1'
#Create and begin a new operation at the specified equipment
path
oper = system.mes.createOperation('Press Door', eqPath)
oper.begin()
#Create new segment
```

```
seg = oper.createSegment('Press Door')
#Assign supplemental equipment resources....
#Add custom properties
#Custom property definition format: {custom_property_name:
[ignition_data_type, value, description, units,
production_visible, required}
cp = {'Wear mm' : ['Int4', 10, 'Amount of wear', 'mm', True, Tr
ue]}
seg.setSupplementalEquipment('Press Die', 'LH Door Die 129',
cp)
#Begin segment
seg.begin()
```

Code Snippet

```
eqPath = '[global]\ABC Inc\California\Pressing\Press 1'
#Create and begin a new operation at the specified equipment
path
oper = system.mes.createOperation('Press Door', eqPath)
oper.begin()
#Create new segment
seg = oper.createSegment('Press Door')
#Assign supplemental equipment resources....
#Set existing custom properties
cp = {'Wear mm' : 11}
seg.setSupplementalEquipment('Press Die', 'LH Door Die 129',
cp)
#Begin segment
seg.begin()
```

update

Description

Update an active segment. If material, personnel, supplemental equipment resources or custom properties change during a production task, then the update script function is used to commit the changes. For resources, these changes are time stamped and save for the segment. This allows for an accurate history of multiple lots, personnel and supplemental equipment that were used during a production task.

Syntax

update()

- Parameters

None

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
#Get the current operation being run at the equipment
oper = system.mes.getCurrentOperation(eqPath)
#Get the active segment running under the operation
segName = oper.getActiveSegmentName()
seg = oper.getActiveSegment(segName)
#Make changes to material resources....
#Make changes personnel resources....
#Make changes supplemental resources....
#Commit the changes
seg.update()
```

All of these script functions require an instance of a MESResponseSegment object. The following code snippets provide examples of how to get an instance of a MESResponseSegment.

Scripting to create an MESResponseSegment

Code Snippet

```
#Create a new segment to be run at the specified equipment
eqPath = '[global]\Dressings Inc\California\Raw Materials\Unload
Station 1'
seg = system.mes.createSegment('Receive Housings')
```

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw Materials\Unload
Station 1'
#Create a new operation at the specified equipment path
oper = system.mes.createOperation('Receive Steel', eqPath)
#Or, If operations are already defined with this equipment, get
the desired operation with the following script.
oper = system.mes.getAvailableOperations(eqPath, 'Receive Steel',
True, True)
#Begin the operation.
oper.begin()
#Create new segment
seg = oper.createSegment('Inspect Steel')
```

Code Snippet

```
eqPath = '[global]\Dressings Inc\California\Raw Materials\Unload
Station 1'
#Get the current operation being run at the equipment
oper = system.mes.getCurrentOperation(eqPath)
#Get the active segment running under the operation
segName = oper.getActiveSegmentName()
seg = oper.getActiveSegment(segName)
```

Events

Besides the common Event Overview, the following events exist for Response Segment objects.

Setting Name	Description
BeforeAutoBegin	The event is fired before the automatic begin of a Response Segment. If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()
BeforeAutoEnd	The event is fired before the automatic end of a Response Segment. If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line:

Setting Name	Description
	event.runDefaultHandler()
BeforeBegin	The event is fired just before a Response Segment object starts.
BeforeEnd	The event is fired just before a Response Segment object ends.
BeginTrace	This event is run every time a Response Segment object starts. This event provides a method to perform tasks when a Response Segment begins. Information about the MES object is passed to the event in a MES Script Event object.
EndTrace	This event is run every time a Response Segment object ends. This event provides a method to perform tasks when a Response Segment ends. Information about the MES object is passed to the event in a MES Script Event object.
New	The event is fired when a new instance of an Response Segment object is created. If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()
SetRecipe	The event is fired when a recipe is set on the Response Segment. If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()
UpdateProgress	This event is run at an interval defined by the Update Event Interval in the Operations Definition. The Enable Update Event setting must also be set to true. This event provides a method to update production counts or other information associated with an active Response Segment. Information about the MES object is passed to the event in a MES Script Event object.

Extended Properties

Setting Name	Description
Begin Date Time	The date and time the operation started. This setting is set when the operation is started, and it will be automatically set to the date and time of the Ignition server.
End Date Time	The date and time the operation ended. This setting is set when the operation is ended, and it will be automatically set to the date and time of the Ignition server.
Operation Segment Reference Type	This setting is automatically set and should not be changed. It will either be set to Operations Segment or Request Segment depending on how the operation was started. If the operation was scheduled prior to being run, then it will equal Request Segment, otherwise it will equal Operations Segment.
Operation Segment Reference UUID	This setting is automatically set and should not be changed. It refers to the UUID of the Operations Segment or Request Segment, depending on how the operation was started. If the operation was scheduled prior to being run, then it will equal the UUID of the Request Segment it is based on, otherwise it will equal the UUID of the Operations Segment it is based on.
Operations Response Reference UUID	This setting is automatically set and should not be changed. It refers to the UUID of the Operations Response this Response Segment is based on.
Running State	This is set by the system and should be changed. The possible values and their meaning is as follows:***Options*** **Idle** - Segment is not in use. **Running** - Segment is currently being used. **Complete** - Segment has completed and is used in the transition from RUNNING to IDLE. **Action Needed** - Some rectifiable error has occured, immediate action should be taken. **Faulted** - An unrecoverable error has occurred in the segment.

The MESResponseSegment also has Lot, Personnel and Equipment Resource Properties based on the AbstractMESComplexProperty Object.

Lot Resource Property

A lot resource property is added for every material resource property that is defined in the Operations Segment or Request Segment that this Response Segment is based on. It represents a lot supplying or lot output from this Response Segment.

During the life of a Response Segment, multiple lot resource properties maybe created for a single material resource property of the Operations Segment. This happens if a lot is changed as maybe the case when a raw material tanks is emptied and changed to another tank.

The setting name is what appears in the MES Object Editor component and the script name is what is used to set or get the value using script. See AbstractMESComplexProperty for details about accessing values using script.

Setting Name	Script Name	Description
Lot Resource Name		This is the name to refer to this lot resource by. Many response segments have multiple lot resources and this is a unique name displayed to the operator, shown in analysis and reports, and also internally used to reference this lot resource.
Lot Reference UUID	LotRefUUID	The UUID of the Material Lot object that this lot resource is linked to.
Lot Reference Sequence	LotRefSequence	The Material Lot object has a lot sequence property that is incremented every time a new Material Lot is created for a given lot number. This makes it unique as an unchanging lot number flows through a manufacturing facility.
Lot Number Source	LotNoSource	This determines the source of the lot number. ***Options*** **Manual** - prompt the operator for the lot number. This is typically used when receiving raw materials or entering a lot number generated by an outside system.

Setting Name	Script Name	Description
		Auto - automatically generated lot number. The internal lot number generator will generate a lot number and assign it automatically for the operator. This option can also be used if a different lot number format is used or lot numbers are provided by another system that is integrated with this system. **In Link** - In cases where the lot number of output of a segment will be the same as the lot number of one of the inputs of the same segment, this setting will tie the two together. Segments can be configured with multiple material inputs and outputs and different lot number links can be configured.
Lot Number Source Link	LotNoSourceLink	If the Lot Number Source setting is set to In Link, then this is the name of the material resource to get the lot number from.
Lot Manual Lot Number	LotManualLotNo	The lot number, usually specified by the operator, to use when the new Material Lot object that is associated with this lot resource is created.
Lot Auto Generate Lot	LotAutoGenerate	If true, a new lot will be generated for the material output of a segment. This is typically only done when receiving material that a lot doesn't already exists.
Lot Material Reference UUID	LotMaterialRefUUID	The UUID of the Material Definition to assign to a new Material Lot when the auto generate lot setting is true.
Lot Begin Date Time	LotBeginDateTime	The date and time that this lot resource started.

Setting Name	Script Name	Description
Lot End Date Time	LotEndDateTime	The date and time that this lot resource ended.
Lot Use	LotUse	This follows the Use setting of the Process Segment. ***Options*** **In** - is used for material feeding into a segment that will be part of the finished goods. **Out** - is used for material feeding out of a segment that is or will be part of the finished goods. **Consumable** - is used for material feeding into a segment that is not part of the finished goods. **By-product** - is used for material feeding out of a segment that is not part of the finished goods.
Lot Equipment Reference Type	LotEquipmentRefType	When a Material Lot is not referenced for this lot resource and a new Material Lot created this setting will define the type of location of the new lot.
Lot Equipment Reference UUID	LotEquipmentRefUUID	When a Material Lot is not referenced for this lot resource and a new Material Lot created this setting is the UUID of the specific location of the new lot.
Lot Segment Reference UUID	LotSegmentRefUUID	This is set to the UUID of the Operations Segment that this Response Segment is based on.
	LotAllocatedQuantity	

Setting Name	Script Name	Description
Lot Allocated Quantity		Currently this is here for reference and is included to align with the ISA-95 standard and will become significant in the next phase of the Track and Trace Module.
Lot Quantity	LotQuantity	The actual quantity for this lot resource. This can be the current quantity at anytime during the life of a Response Segment, but will equal the final production quantity when this lot resource is finalized.
Lot Quantity Source	LotQuantitySource	This setting determines the source of the quantity for this material resource. ***Options*** **Auto** - Obtain the quantity from the automatic production counters defined for the associated equipment. The associated equipment may change if the Lot Equipment Reference setting is set to a Material Class and the specific equipment is not known until the segment is started for production. **Link** - This option allows the quantity to come from an input or output material resource of this segment. This eliminates the need to type in the quantity multiple times if they will always be the same as another material resource. **Manual** - The operator will be prompted for the quantity. The quantity must be entered before the segment is ended. **Split** - For segments that are splitting a lot into two or more streams, as is the case of separating good from bad product, this option can be used. It is used by having two or more material resources, that are segment outputs, linked to the same material resource. When the segment is ended, the system will ensure that the sum of the quantities of the linking material resources equal that of the linked material resources.

Setting Name	Script Name	Description
		Sublots - The quantity will be automatically set based on the number of Material Sublot items belonging to the Material Lot. If sublots are used, then serial numbers, or other unique identification number, can be assigned to each sublot item. For example, a Material Lot of batteries maybe have 25 individual batteries each with a serial number and each with their own test results. The quantity of the Material Lot will match the number of Material Sublot items of the Material Lot. Or, the number of batteries in the lot. **Combine** - For segments that are combining two or more lots into one streams, as is the case of joining goods after tests are done to only a portion of a lot, this option can be used. It is used by having two or more material resources, that are segment inputs, linked to the same material resource output. When the segment is ended, the system will sum of the quantities of the linked material resources to that of the linking material resources.
Lot Quantity Source Link	LotQuantitySourceLink	This is used when the Quantity Source setting is set to Link, Split or Combine. It is the name of the material resource to link to this segment.
Lot Quantity Units	LotQuantityUnits	The units for the Lot Quantity value.
Lot Property Status	LotPropertyStatus	The status of the lot resource. The system changes this value through the life cycle of a lot resource property. ***Options*** **Beginning** - It is a new lot resource property. **Active** - The lot resource is currently active.

Setting Name	Script Name	Description
		Ending - The lot resource is ending and information will be finalized for it. **Complete** - The lot resource is complete and is no longer begin used by the Response Segment. **Update_Sublots** - The list of associated Material Sublot objects has changed and are being updated.
Lot Final Lot Status	LotFinalLotStatus	When a segment is started, the status of the Material Lots will be set to Active. When the segment is ended or a new lot is used for the material resource, the status will be set to Complete. Optionally, the value of this setting can be used instead of the default Complete. Please note, the Active status while the lot is active cannot be changed. This is useful for setting a lot to Hold, In Process or anything that can be used to filter lots or sublots.
Lot Enable Sublots	LotEnableSublots	If this setting is selected, then sublot support will be enabled for the material resource. If sublots are used, then serial numbers, or other unique identification number, can be assigned to each sublot item. For example, a Material Lot of batteries maybe have 25 individual batteries each with a serial number and each with their own test results.

Personnel Resource Property

A personnel resource property is added for every personnel resource property that is defined in the Operations Segment or Request Segment that this Response Segment is based on. It represents an actual person involved in the Response Segment. The setting name is what appears in the MES Object Editor component and the script name is what is used to set or get the value using script. See AbstractMESComplexProperty for details about accessing values using script.

Setting Name	Script Name	Description
Personnel Name		This is the name to refer to this personnel resource by. Some process segments have multiple personnel resources and this is a unique name displayed to the operator, shown in analysis and reports, and also internally used to reference this personnel resource.
Personnel Reference	PersonnelRef	This can be set to a Personnel Class or a Person. By setting this to Personnel Class will cause the operator to be prompted for the specific Person for this personnel resource. If set to a Person, then the selection will be automatically selected.
MES Object Name		Depending on the setting of the type, Personnel Class or Person options will show. When Process Segments or Operations Segments are inherited from a Process Segment, then the options that show for the Personnel Reference setting will be limited by the parent settings. For example: If Unload Operator Personnel class is selected for a Process Segment and a new child Process Segment is created from it, then the only options will be limited to the Unload Operator Class and any child of it.
	PersonnelRefUUID	UUID of the selected Personnel Class or Person.
Units	PersonnelUnits	This specifies the units for the quantity setting.
Use	PersonnelUse	Currently this is here for reference and is included to align with the ISA-95 standard and will become significant in the next phase of the Track and Trace Module.
Quantity	PersonnelQuanity	Currently this is here for reference and is included to align with the ISA-95 standard and will become significant in the next phase of the Track and Trace Module.

Equipment Resource Property

An equipment resource property is added for every equipment resource property that is defined in the Operations Segment or Request Segment that this Response Segment is based on. It represents an actual equipment involved in the Response Segment. The setting name is what appears in the MES Object Editor component and the script name is what is used to set or get the value using script. See AbstractMESComplexProperty for details about accessing values using script.

Setting Name	Script Name	Description
Equipment Resource Name		This is the name to refer to this equipment resource by. Some process segments may have multiple equipment resources and this is a unique name displayed to the operator, shown in analysis and reports, and also internally used to reference this equipment resource.
Equipment Reference	EquipmentRef	This can be set to a Equipment Class or a Equipment, Line, Line Cell, Line Cell Group or Storage Unit. By setting this to Equipment Class will cause the operator to be prompted for the specific equipment for this equipment resource. If set to a specific equipment item, then the selection will be automatically selected.
MES Object Name		Depending on the setting of the type, Material Class or specific equipment item options will show. When Process Segments or Operations Segments are inherited from a Process Segment, then the options that show for the Equipment Reference setting will be limited by the parent settings. For example: If Unload Stations class is selected for a Process Segment and a new child Process Segment is created from it, then the only options will be limited to the Unload Stations Class and any child of it.
	EquipmentRefUUID	UUID of the Equipment Class or Equipment item.
Units	EquipmentUnits	This specifies the units for the quantity setting.
Use	EquipmentUse	

Setting Name	Script Name	Description
		Currently this is here for reference and is included to align with the ISA-95 standard and will become significant in the next phase of the Track and Trace Module.
Quantity	EquipmentQuanity	Currently this is here for reference and is included to align with the ISA-95 standard and will become significant in the next phase of the Track and Trace Module

9.6.3 OEE Objects

Abstract Value Item Info

Object Description

The AbstractValueItemInfo object holds the information of an analysis value item that can be a value source or calculator.

Scripting Functions

The following function can be used to get Abstract Value Item Info object.

system.mes.analysis.getDataPointOptions(groupFilter, itemFilter)

Description

Return data point options that can be used when executing analysis.

Syntax

system.mes.analysis.getDataPointOptions(groupFilter, itemFilter)

- Parameters

String groupFilter - A filter to limit the data point options returned to one or more groups. Multiple groups can be specified by separating them with commas. The wildcard * is accepted.

String itemFilter - A filter to limit the data point options returned to one or more items. Multiple data point items can be specified by separating them with commas. The wildcard * is accepted.

- Returns

List<AbstractValueItemInfo> - Returns a map (a key-value pair) containing the filter group path as the key and a list of AbstractValueItemInfo objects as the value. See AbstractValueItemInfo object documentation for details.

- Scope

All

Code Examples

Code Snippet

```
##Get a list of data point options for the Equipment group:
list = system.mes.analysis.getDataPointOptions('Equipment', '*'
)
for item in list:
    for x in list[item]:
        print item, '::', x.getName()
```

Output

```
Equipment :: Product Code
Equipment :: Work Order
Equipment :: Is Key Cell
Equipment :: Equipment Type
Equipment :: Equipment Name
Equipment :: Operation UUID
Equipment :: Equipment Path
Equipment :: Equipment Cell Order
Equipment :: Rate Period
```

system.mes.analysis.getFilterOptions(groupFilter, itemFilter)

Description

Return filter options that can be used when executing analysis.

Syntax

system.mes.analysis.getFilterOptions(groupFilter, itemFilter)

- Parameters

String groupFilter - A filter to limit the filter options returned to one or more groups. Multiple groups can be specified by separating them with commas. The wildcard * is accepted.

String itemFilter - A filter to limit the group by options returned to one or more items. Multiple filter items can be specified by separating them with commas. The wildcard * is accepted.

- Returns

List <AbstractValueItemInfo> - Returns a map (a key-value pair) containing the filter group path as the key and a list of AbstractValueItemInfo objects as the value. See AbstractValueItemInfo object documentation for details.

- Scope

All

Code Examples

Code Snippet

```
##Get a list of filter options from the OEE group:
list = system.mes.analysis.getFilterOptions('OEE', '*')
for item in list:
    for x in list[item]:
        print item, '::', x.getName()
```

Output

```
OEE :: OEE Infeed Count Equipment Path
OEE :: Target Changeover Time
OEE :: OEE
OEE :: Standard Rate
OEE :: Elapsed Time
```

```
OEE :: OEE Outfeed Count Equipment Path
OEE :: Schedule Rate
```

system.mes.analysis.getGroupByOptions(groupFilter, itemFilter)

Description

Return group-by options that can be used when executing analysis.

Syntax

system.mes.analysis.getGroupByOptions(groupFilter, itemFilter)

- Parameters

String groupFilter - A filter to limit the group-by options returned to one or more groups. Multiple groups can be specified by separating them with commas.

String itemFilter - A filter to limit the group-by options returned to one or more items. Multiple group-by items can be specified by separating them with commas.

- Returns

List<AbstractValueItemInfo> - Returns a map (a key-value pair) containing the filter group path as the key and a list of AbstractValueItemInfo objects as the value. See AbstractValueItemInfo object documentation for details.

- Scope

All

system.mes.analysis.getOrderByOptions(groupFilter, itemFilter)

Description

Return order by options that can be used when executing analysis.

Syntax

system.mes.analysis.getOrderByOptions(groupFilter, itemFilter)

- Parameters

String groupFilter - A filter to limit the order by options returned to one or more groups. Multiple groups can be specified by separating them with commas.

String itemFilter - A filter to limit the order by options returned to one or more items. Multiple order by items can be specified by separating them with commas.

- Returns

List<AbstractValueItemInfo> - Returns a map containing the filter group path in the key and a list of AbstractValueItemInfo objects in the value. See AbstractValueItemInfo object documentation for details.

- Scope

All

Methods

The following methods exist for the Abstract Value Item Info object.

getDescription()

Description

Returns the description for the Abstract Value Item object.

Syntax

getDescription()

- Parameters

None

- Returns

String - The description for this AbstractValueItem object.

- Scope

All

isBoolean()

Description

Returns True, if it is a boolean value.

Syntax

isBoolean()

- Parameters

None

- Returns

Boolean - True, if it is a boolean.

- Scope

All

isDateTime()

Description

Returns True if this Abstract Value Item object holds a date and time value.

Syntax

isDateTime()

- Parameters

None

- Returns

Boolean - True if the object holds a date and time.

- Scope

All

isDouble()

Description

Returns True if the object holds a double value.

Syntax

isDouble()

- Parameters

None

- Returns

Boolean - True, if the object is a double.

- Scope

All

isInteger()

Description

Returns True if this object is an integer.

Syntax

isInteger()

- Parameters

None

- Returns

Boolean - True, if the object is an integer.

- Scope

All

isNumeric()

Description

Returns True if this object holds a numeric value.

Syntax

isNumeric()

- Parameters

None

- Returns

Boolean - True, if it is a numeric value.

- Scope

All

isString()

Description

Returns True if this object holds a string.

Syntax

isString()

- Parameters

None

- Returns

Boolean - True, if the object is a string.

- Scope

All

Analysis Parameter Property

Base Object

The Analysis Parameter Property is derived from the AbstractMESComplexProperty and inherits all the exposed properties, methods and events for that object.

Methods

Beside the common AbstractMESComplexProperty methods, the following methods exist for the Analysis Parameter Property object .

getParameterDataType()

Description

Returns the data type of the parameter.

Syntax

getParameterDataType()

- Parameters

None

- Returns

String type - The data type of the parameter.

- Scope

All

setParameterDataType(type)

Description

Sets the data type of the parameter.

Syntax

setParameterDataType(type)

- Parameters

String type - The data type to set for the parameter.

- Returns

Nothing

- Scope

All

Events

Besides the common MES object events, no other events exist for the Analysis Parameter Property object.

Properties

Property values can be accessed and changed for an object by using the getPropertyValue() and setPropertyValue() method.

Example

```
obj = system.mes.MESObject('')    #Return a MES object
print obj.getPropertyValue('')
```

Besides the common core properties, the following properties exist for Analysis Parameter Property object.

Setting Name	R/W	Description
AnalysisParameterDataTypeProperty	Read /Write	The data type property for Analysis Parameter Property object.

Analysis Security Property

Base Object

The Analysis Security Property object is derived from the AbstractMESComplexProperty and inherits all the exposed properties, methods and events for that object.

Methods

Beside the common MESAbstractObject methods, the following methods exist for the Analysis Security Property object.

getCanExecute()

Description

Gets the boolean assigned for the can execute property.

Syntax

getCanExecute()

- Parameters

None

- Returns

Boolean - True if execution of analysis security property is allowed and False otherwise.

- Scope

All

getCanModify()

Description

Gets the boolean assigned for the can modify property.

Syntax

getCanModify()

- Parameters

None

- Returns

Boolean - True if modification of analysis security property is allowed and False otherwise.

- Scope

All

setCanExecute(canExecute)

Description

Sets the can execute property for saved analysis. If this property is set to True, then the saved analysis can be executed.

Syntax

setCanExecute(canExecute)

- Parameters

Boolean canExecute - Set to True to allow execution and False otherwise.

- Returns

Nothing

- Scope

All

setCanModify(canModify)

Description

Sets the can modify property for saved analysis. If this property is set to True, then the saved analysis can be modified.

Syntax

setCanModify(canModify)

- Parameters

Boolean canModify - Set to True to allow modification and False otherwise.

- Returns

Nothing

- Scope

All

Events

Besides the common MES object events, no other events exist for the Analysis Security Property.

Properties

Property values can be accessed and changed for an object by using the getPropertyValue() and setPropertyValue() method.

Example

```
obj = system.mes.MESObject('')    #Return a MES object
print obj.getPropertyValue('')
```

Besides the common core properties, the following properties exist for Analysis Security Property object.

Setting Name	R/W	Description
AnalysisSecurityCanExecuteProperty	Read /Write	If this property is set to True, then the saved analysis can be executed.

AnalysisSecurityCanModifyProperty	Read /Write	If this property is set to True, then the saved analysis can be modified.

MES Equipment Mode

Equipment Modes support tracking of line idle time, changeover, production, preventative maintenance, training, testing and even custom modes. Default equipment modes are automatically defined as shown below, but if they do not meet your requirements, they can be overridden.

Mode	Code	Include in OEE	Include in Counts
Unknown	0	false	false
Production	1	true	true
Idle	2	false	false
Changeover	3	false	false
Maintenance	4	false	false
Other	5	false	false
Disabled	6	false	false

This provides two important capabilities. The first is the ability to analyze the duration of time spent in each mode or mode category to clearly show the equipment utilization. The second is complete control of which mode to include in OEE and production counts.

Methods

getIncludeInOEE()

Description

Checks whether the include OEE property is enabled and returns the corresponding boolean.

Syntax

getIncludeInOEE()

- Parameters

None

- Returns

Boolean - True if include OEE property is enabled and False otherwise.

getIncludeProductionCounts()

Description

Checks whether the include production counts property is enabled and returns the corresponding boolean.

Syntax

getIncludeProductionCounts()

- Parameters

None

- Returns

Boolean - True if include production counts property is enabled and False otherwise.

getMESObjectType()

Description

Gets the MES object type associated with this equipment mode.

Syntax

getMESObjectType()

- Parameters

None

- Returns

MESObjectTypes type - The MES object type associated with this equipment mode.

getModeCode()

Description

Returns the code (an Integer) that represents this equipment mode.

Syntax

getModeCode()

- Parameters

None

- Returns

Integer code - The integer that represents this equipment mode.

getModeType()

Description

Gets the type of equipment mode.

Syntax

getModeType()

- Parameters

None

- Returns

EquipmentModeTypes modeType - The type of the equipment mode.

getModeTypeName()

Description

Gets the equipment mode type.

Syntax

getModeTypeName()

- Parameters

None

- Returns

String modeType - The name of the equipment mode type.

Events

Besides the common MES object events, the following events exist for Equipment Mode object.

Setting Name	Description
New	The event is fired when a new instance of a Equipment Mode object is created. If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()

MES Equipment Mode Class

Object Description

This object holds information about the Equipment Mode Classes. New equipment mode classes can be created using the OEE Equipment Manager component.

Base Object

The MES Equipment Mode Class object is derived from the MESAbstractObject and inherits all the exposed properties, methods and events for that object.

MES Equipment State

In OEE 2.0, Downtime Events are now replaced with Equipment States that have a configurable State Type . The State Type provides the ability to logically group equipment states together.

Default State Types

- IDLE
- STARVED
- RUNNING
- BACKUP
- DISABLED
- DOWN PLANNED
- DOWN UNPLANNED

This provides the following benefits…

- Raw equipment status coming from the PLC maintains its finite state machine which is captured and stored and allows for cycle time analysis etc.
- PLC status for LOADING, HEATING, PURGING etc. can be grouped and considered as RUNNING for OEE.

Methods

getStateCode()

Description

Returns the code (an integer) that represents this equipment state.

Syntax

getStateCode()

- Parameters

None

- Returns

Integer - The integer code that is associated with this equipment state.

getStateOverrideScope()

Description

Returns the override scope of this equipment state.

Syntax

getStateOverrideScope()

- Parameters

None

- Returns

String scope - The override scope associated with this equipment state.

getStateOverrideSetting()

Description

Returns the override setting of this equipment state.

Syntax

getStateOverrideSetting()

- Parameters

None

- Returns

String setting - The override setting associated with this equipment state.

getStateShortStopThreshold()

Description

Returns the duration (in seconds) that a state will be considered as a short stop (if less than). If state duration is longer, then it will be considered a downtime event, or planned downtime event or other.

Syntax

getStateShortStopThreshold()

- Parameters

None

- Returns

Integer threshold - T he duration (in seconds) that a state will be considered as a short stop.

getStateType()

Description

Returns the type of the equipment state.

Syntax

getStateType()

- Parameters

None

- Returns

EquipmentStateTypes type - The type of this equipment state.

getStateTypeName()

Description

Returns the name of the equipment state type.

Syntax

getStateTypeName()

- Parameters

None

- Returns

EquipmentStateTypes type - The name of the equipment state mode.

Events

Besides the common MES object events, the following events exist for Equipment State objects.

Setting Name	Description
New	The event is fired when a new instance of a Equipment State object is created. If no script is entered, then the default handler will be executed. To execute the default handler from within the script entered here, add a script line: event.runDefaultHandler()

MES Equipment State Class

Object Description

This object holds information about the Equipment State Classes. New equipment state classes can be created using OEE Equipment Manager component.

Base Object

The MES Equipment State Class object is derived from the MESAbstractObject and inherits all the exposed properties, methods and events for that object.

MES Material Root

Object Description

The Material Root is the base class for all OEE 2.0 Material Classes and Definitions. All materials utilized in OEE 2.0 operations (e.g. production runs, etc.) are children of the Material Root. There is only one Material Root object per Ignition Gateway instance. The Material Root object is created automatically by the OEE 2.0 module.

Base Object

The MES Material Root object is derived from the MESAbstractObject and inherits all the exposed properties, methods and events for that object.

Methods

getIdealStandardCount()

Description

Gets the ideal standard count of this OEE counter.

Syntax

getIdealStandardCount()

- Parameters

None

- Returns

int idealStandardCount - The ideal standard count of this OEE counter.

- Scope

All

getInfeedCount()

Description

Gets the infeed count.

Syntax

getInfeedCount()

- Parameters

None

- Returns

int infeedCount - The infeed count.

- Scope

All

getName()

Description

Gets name of the OEE counter.

Syntax

getName()

- Parameters

None

- Returns

String name - The name of this OEE counter.

- Scope

All

getPackageCount()

Description

Gets the package count.

Syntax

getPackageCount()

- Parameters

None

- Returns

Double packageCount - The package count.

- Scope

All

getPath()

Description

Gets path of the OEE counter.

Syntax

getPath()

- Parameters

None

- Returns

String path - The path of this OEE counter.

- Scope

All

getProductionCount()

Description

Gets the production count.

Syntax

getProductionCount()

- Parameters

None

- Returns

int productionCount - The production count.

- Scope

All

getStandardCount()

Description

Gets the standard count of this OEE counter.

Syntax

getStandardCount()

- Parameters

None

- Returns

int standardCount - The standard count of this OEE counter.

- Scope

All

getStandardRate()

Description

Gets the standard rate.

Syntax

getStandardRate()

- Parameters

None

- Returns

Double standardRate - The standard rate.

- Scope

All

getStandardRatePeriod()

Description

Gets standard rate period of the OEE counter.

Syntax

getStandardRatePeriod()

- Parameters

None

- Returns

String standardRatePeriod - The standard rate period of this OEE counter.

- Scope

All

getStandardVariance()

Description

Gets the standard variance of this OEE counter.

Syntax

getStandardVariance()

- Parameters

None

- Returns

int standardVariance - The standard variance of this OEE counter.

- Scope

All

getTargetCount()

Description

Gets the target count.

Syntax

getTargetCount()

- Parameters

None

- Returns

int targetCount - The target count.

- Scope

All

getTargetVariance()

Description

Gets the target variance of this OEE counter.

Syntax

getTargetVariance()

- Parameters

None

- Returns

int targetVariance - The target variance of this OEE counter.

- Scope

All

getWasteCount()

Description

Gets the waste count of this OEE counter.

Syntax

getWasteCount()

- Parameters

None

- Returns

int wasteCount - The waste count of this OEE counter.

- Scope

All

setIdealStandardCount(idealStandardCount)

Description

Sets the ideal standard count for this OEE counter.

Syntax

setIdealStandardCount(idealStandardCount)

- Parameters

int idealStandardCount - The ideal standard count to set for.

- Returns

Nothing

- Scope

All

setStandardCount(standardCount)

Description

Sets the standard count for this OEE counter.

Syntax

setStandardCount(standardCount)

- Parameters

int standardCount - The standard count to set for.

- Returns

Nothing

- Scope

All

setStandardVariance(standardVariance)

Description

Sets the standard variance for this OEE counter.

Syntax

setStandardVariance(standardVariance)

- Parameters

int standardVariance - The standard variance to set for.

- Returns

Nothing

- Scope

All

setTargetCount(targetCount)

Description

Sets the target count for this OEE counter.

Syntax

setTargetCount(targetCount)

- Parameters

int targetCount - The count to set as a target for.

- Returns

Nothing

- Scope

All

setTargetVariance(targetVariance)

Description

Sets the target variance for this OEE counter.

Syntax

setTargetVariance(targetVariance)

- Parameters

int targetVariance - The target variance to set for.

- Returns

Nothing

- Scope

All

setWasteCount(wasteCount)

Description

Sets the waste count for this OEE counter.

Syntax

setWasteCount(wasteCount)

- Parameters

int wasteCount - The waste count to set for.

- Returns

Nothing

- Scope

All

9.6.4 Schedule Objects

Properties:

getScheduleCategory()

Description

Returns the category of an operations schedule object.

Syntax

getScheduleCategory()

- Parameters

None

- Returns

String scheduleCategory - The category of the operations schedule object.

- Scope

All

getScheduleDurationSec()

Description

Gets the duration of the schedule in seconds.

Syntax

getScheduleDurationSec()

- Parameters

None

- Returns

Integer scheduleDuration - The duration of schedule in seconds.

- Scope

All

getScheduleProductionCount()

Description

Returns the production count for this schedule.

Syntax

getScheduleProductionCount()

- Parameters

None

- Returns

Integer productionCount -The production count for this schedule.

- Scope

All

getSchedulePublishDate()

Description

Returns date and time at which schedule was published. This setting is set when the operation is started, and it will be automatically set to the date and time of the Ignition server.

Syntax

getSchedulePublishDate()

- Parameters

None

- Returns

Date - The date and time the schedule was published.

- Scope

All

getScheduleType()

Description

Gets the schedule type.

Syntax

getScheduleType()

- Parameters

None

- Returns

String type - The type of this schedule.

- Scope

All

setScheduleCategory(scheduleCategory)

Description

Sets the category of an operations schedule object. Options for the toCategory parameter is either 'Hold' or 'Record Scheduled'.

Syntax

setScheduleCategory(scheduleCategory)

- Parameters

String scheduleCategory - The category of the operations schedule object.

- Returns

Nothing

- Scope

All

setScheduleDurationSec(scheduleDuration)

Description

Sets the duration of the schedule in seconds.

Syntax

setScheduleDurationSec(scheduleDuration)

- Parameters

Integer scheduleDuration - The duration in seconds to set for.

- Returns

Nothing

- Scope

All

setScheduleProductionCount(productionCount)

Description

Sets the production count for this schedule.

Syntax

setScheduleProductionCount(productionCount)

- Parameters

Integer productionCount -The production count to set for this schedule.

- Returns

Nothing

- Scope

All

setSchedulePublishDate(schedulePublishDate)

Description

Sets date and time at which schedule was published. This setting is set when the operation is started, and it will be automatically set to the date and time of the Ignition server.

Syntax

setSchedulePublishDate(schedulePublishDate)

- Parameters

Date schedulePublishDate - The date to set for.

- Returns

Nothing

- Scope

All

setScheduleType(scheduleType)

Description

Sets the schedule type. This can be set to Held or Active.

Syntax

setScheduleType(scheduleType)

- Parameters

String scheduleType - The type to set for.

- Returns

Nothing

- Scope

All

MES Schedule Entry

The Schedule Entry is an object associated with MES Schedule View component and is passed in the userMenuItemClicked event.

Properties:

determinePriorityState(state1, state2)

Description

Compares the priorities of the states and returns the state with highest priority.

Syntax

determinePriorityState(state1, state2)

- Parameters

int state1 - The first state to compare.

Integer state2 - The second state to compare.

- Returns

The state with highest priority.

- Scope

All

getActualEndDate()

Description

Returns the actual end date for this schedule.

Syntax

getActualEndDate()

- Parameters

None

- Returns

Date actualEndDate - The actual end date for the schedule.

- Scope

All

getActualStartDate()

Description

Returns the actual start date for this schedule.

Syntax

getActualStartDate()

- Parameters

None

- Returns

Date actualStartDate - The actual start date of the schedule.

- Scope

All

getCategory()

Description

Gets the category of this schedule entry.

Syntax

getCategory()

- Parameters

None

- Returns

String category - The category of this schedule entry.

- Scope

All

getDefaultToolTipText()

Description

Returns the tool tip text with details of schedule start, end and progress.

Syntax

getDefaultToolTipText()

- Parameters

None

- Returns

The tool tip text for this schedule entry.

- Scope

All

getLabel()

Description

Gets the label for this schedule.

Syntax

getLabel()

- Parameters

None

- Returns

String label - The label corresponding to this schedule entry.

- Scope

All

getMESOperationsRequestLink()

Description

Returns the link to this operations request object.

Syntax

getMESOperationsRequestLink()

- Parameters

None

- Returns

MESObjectLink mesOperationsRequestLink - The link to the operations request object.

- Scope

All

getMESOperationsResponseLink()

Description

Returns the MES link to the operations response object.

Syntax

getMESOperationsResponseLink()

- Parameters

None

- Returns

MESObjectLink mesOperationsResponseLink - The link corresponding to operations response object.

- Scope

All

getMESOperationsScheduleLink()

Description

Returns the MES object link to the operations schedule object.

Syntax

getMESOperationsScheduleLink()

- Parameters

None

- Returns

MESObjectLink - The link to the operations schedule object.

- Scope

All

getOriginalState()

Description

Returns the original state of this schedule entry.

Syntax

getOriginalState()

- Parameters

None

- Returns

Integer originalState - The state at which the schedule started.

- Scope

All

getProgressPercent()

Description

Gets the progress of work done in percentage for this schedule.

Syntax

getProgressPercent()

- Parameters

None

- Returns

double progressPercent - The percentage of work done for this schedule entry.

- Scope

All

getScheduledEndDate()

Description

Returns the end date for this schedule.

Syntax

getScheduledEndDate()

- Parameters

None

- Returns

Date scheduledEndDate - The end date for this schedule.

- Scope

All

getScheduledStartDate()

Description

Returns the start date of the schedule.

Syntax

getScheduledStartDate()

- Parameters

None

- Returns

Date scheduledStartDate - The date at which this schedule starts.

- Scope

All

getState()

Description

Gets the state of this schedule entry.

Syntax

getState()

- Parameters

None

- Returns

Integer state - The state of this schedule entry.

- Scope

All

hasMESOperationsResponseLink()

Description

Checks the existence of an MES link to the operations response object.

Syntax

hasMESOperationsResponseLink()

- Parameters

None

- Returns

boolean mesOperationsResponseLink - True, if there is any link corresponding to operations response object and False otherwise.

- Scope

All

hasMESOperationsScheduleLink()

Description

Checks for any MES object link to the operations schedule object.

Syntax

hasMESOperationsScheduleLink()

- Parameters

None

- Returns

boolean - True, If there exist an MES object link for the operations schedule and False otherwise.

- Scope

All

hasState()

Description

Checks if there is any state defined for this schedule entry.

Syntax

hasState()

- Parameters

None

- Returns

boolean state - True if there is any state associated with this schedule entry and False otherwise.

- Scope

All

resetState()

Description

Resets the schedule entry to its original state.

Syntax

resetState()

- Parameters

None

- Returns

Nothing

- Scope

All

setActualEndDate(actualEndDate)

Description

Sets the actual end date for this schedule.

Syntax

setActualEndDate(actualEndDate)

- Parameters

Date actualEndDate - The actual end date to set for.

- Returns

Nothing

- Scope

All

setActualStartDate(actualStartDate)

Description

Sets the actual start date for this schedule.

Syntax

setActualStartDate(actualStartDate)

- Parameters

Date actualStartDate - The actual start date for the schedule.

- Returns

Nothing

- Scope

All

setCategory(category)

Description

Sets the category of this schedule entry.

Syntax

setCategory(category)

- Parameters

String category - The category to set for.

- Returns

Nothing

- Scope

All

setLabel(label)

Description

Sets the label for this schedule.

Syntax

setLabel(label)

- Parameters

String label - The label to set for this schedule entry.

- Returns

Nothing

- Scope

All

setMESOperationsRequestLink(MESObjectLink mesOperationsRequestLink)

Description

Sets the link to this operations request object.

Syntax

setMESOperationsRequestLink(mesOperationsRequestLink)

- Parameters

MESObjectLink mesOperationsRequestLink - The link to the operations request object.

- Returns

Nothing

- Scope

All

setMESOperationsResponseLink(mesOperationsResponseLink)

Description

Sets the MES link to the operations response object.

Syntax

setMESOperationsResponseLink(mesOperationsResponseLink)

- Parameters

MESObjectLink mesOperationsResponseLink - The link corresponding to operations response object to set for.

- Returns

Nothing

- Scope

All

setMESOperationsScheduleLink(mesOperationsScheduleLink)

Description

Sets the MES object link to the operations schedule object.

Syntax

setMESOperationsScheduleLink(mesOperationsScheduleLink)

- Parameters

MESObjectLink mesOperationsScheduleLink - The operations schedule link to set for.

- Returns

Nothing

- Scope

All

setProgressPercent(progressPercent)

Description

Sets the percentage of work to be done for this schedule.

Syntax

setProgressPercent(progressPercent)

- Parameters

double progressPercent - The percentage of work to set for.

- Returns

None

- Scope

All

setScheduledEndDate(endDate)

Description

Sets the end date for this schedule.

Syntax

setScheduledEndDate(endDate)

- Parameters

Date scheduledEndDate - The end date to set for.

- Returns

Nothing

- Scope

All

setScheduledStartDate(startDate)

Description

Sets the start date for this schedule.

Syntax

setScheduledStartDate(startDate)

- Parameters

Date startDate - The date to start the schedule for.

- Returns

Nothing

- Scope

All

setState(state)

Description

Sets the state for this schedule entry.

Syntax

setState(state)

- Parameters

boolean state - The state to set for.

- Returns

Nothing

- Scope

All

9.6.5 SPC Objects

SPC Object Types

The SPC Module has script functions and events that use various objects. The following sections provide documentation of the methods and properties associated with these various objects.

Anderson Darling Test

The event is created to calculate an Anderson-Darling test. This test will determine if a data set comes from a specified distribution, in our case, the normal distribution. The test makes use of the cumulative distribution function.

Brief discussion on the variables used for AD test

The Anderson-Darling statistic **AD** is given by the following formula:

where n = sample size, F(X) = cumulative distribution function for the specified distribution and i = the ith sample when the data is sorted in ascending order.

The summation portion of the equation is given below, the summation term **S** in the Anderson-Darling equation:

anderson-darling summation term

The value of AD needs to be adjusted for small sample sizes. The adjusted AD value is given by:

adjusted anderson-darling equation

The p -value is defined as the probability of obtaining a result equal to or "more extreme" than what was actually observed, when the null hypothesis is true. For typical analysis, using the standard = 0.05 cutoff, the null hypothesis is rejected when $p < .05$ and not rejected when $p > .05$. The p -value does not in itself support reasoning about the probabilities of hypotheses but is only a tool for deciding whether to reject the null hypothesis: -wikipedia

Properties:

getAd()

Description

Returns the AD.

Syntax

getAd()

- Parameters

None

- Returns

Double ad - The AD.

- Scope

All

getAdStar()

Description

Returns the AD*.

Syntax

getAdStar()

- Parameters

None

- Returns

Double adStar - The AD*.

- Scope

All

getData()

Description

Returns the SPC data.

Syntax

getData()

- Parameters

None

- Returns

AnalysisDataset data - The SPC data.

- Scope

All

getMean()

Description

Returns the mean.

Syntax

getMean()

- Parameters

None

- Returns

Double mean - The mean.

- Scope

All

getMeasurementCount()

Description

Returns the measurement count.

Syntax

getMeasurementCount()

- Parameters

None

- Returns

int measurementCount - The measurement count.

- Scope

All

getPValue()

Description

Get the p-value for the test statistics. If the p-value is less than a chosen alpha (usually **0.05** or 0.10), then reject the null hypothesis that the data come from that distribution.

Syntax

getPValue()

- Parameters

None

- Returns

Double pValue - The p-value of the Anderson Darling Test.

- Scope

All

getS()

Description

Returns the S value in the Anderson Darling Test.

Syntax

getS()

- Parameters

None

- Returns

Double s - The s of the Anderson Darling Test.

- Scope

All

getStandardDeviation()

Description

Returns the actual standard deviation.

Syntax

getStandardDeviation()

- Parameters

None

- Returns

Double standardDeviation - The standard deviation.

- Scope

All

setAd(ad)

Description

Sets the AD.

Syntax

setAd(ad)

- Parameters

Double ad - The AD.

- Returns

Nothing

- Scope

All

setAdStar(adStar)

Description

Sets the AD*.

Syntax

setAdStar(adStar)

- Parameters

Double data - The AD*.

- Returns

Nothing

- Scope

All

setMean(mean)

Description

Sets the mean.

Syntax

setMean(mean)

- Parameters

Double mean - The mean to set for.

- Returns

Nothing

- Scope

All

setPValue(pValue)

Description

Sets the p-value.

Syntax

setPValue(pValue)

- Parameters

Double pValue - The p-value to set for.

- Returns

Nothing

- Scope

All

setS(s)

Description

Sets the S.

Syntax

setS(s)

- Parameters

Double s - The s.

- Returns

Nothing

- Scope

All

setStandardDeviation(standardDeviation)

Description

Sets the actual standard deviation.

Syntax

setStandardDeviation(standardDeviation)

- Parameters

Double standardDeviation - The standard deviation.

- Returns

Nothing

- Scope

All

Attribute Data Type

The attribute data type object contains the available data types of a sample attribute.

Available data types:

INTEGER

Attribute can contain positive or negative numeric values with no fractions. It has a minimum value of -2,147,483,648 and a maximum value of 2,147,483,647 (inclusive).

REAL

Attribute can contain double-precision 64-bit IEEE 754 floating point values.

BOOLEAN

Attribute can contain a true or false value.

INSPECTED_COUNT

Attribute can contain a counting number (1, 2, 3, 4, …) and represents a number of items inspected for a attribute samples. This attribute data type is recognized and required by the p, np, c and u control charts.

NONCONFORMING_COUNT

Attribute can contain a counting number (1, 2, 3, 4, …) and represents a number of nonconforming items (defective items) for a attribute samples. This attribute data type is recognized and required by the p and np control charts.

NONCONFORMITY_COUNT

Attribute can contain a counting number (1, 2, 3, 4, …) and represents the number of nonconformities items that have (deformities) for a attribute samples. This attribute data type is recognized and required by the c and u control charts.

Properties:

convert(attrValue)

Description

Returns value in the true java data type for the type of data this attribute data type represents.

Syntax

convert(attrValue)

- Parameters

Object attrValue - The attribute value to be converted.

- Returns

The value in java data type.

dataTypeToType(dataType)

Description

Return a reference to a SPC attribute data type from an Ignition data type. For example, an Ignition data type of Float8 will be a attribute data type of Real.

Syntax

dataTypeToType(dataType)

- Parameters

DataType dataType - Ignition DataType reference that represents the type of data for a sample attribute.

- Returns

AttributeDataType - A reference to a AttributeDataType value that matches.

getJavaType()

Description

Returns the java data type for this attribute data type.

Syntax

getJavaType()

- Parameters

None

- Returns

The java data type corresponding to this attribute data type.

getText()

Description

Returns the user friendly localized text for the attribute data type.

Syntax

getText()

- Parameters

None

- Returns

The localized text for this attribute data type.

intToType(ordinal)

Description

Return a reference to a SPC attribute data type from the ordinal value.

Syntax

intToType(ordinal)

- Parameters

int ordinal - A valid AttributeDataType ordinal value.

- Returns

AttributeDataType - A reference to a AttributeDataType value that matches.

isLogical()

Description

Returns true if the attribute data type is boolean.

Syntax

isLogical()

- Parameters

None

- Returns

True, if the attribute data type is boolean.

isNumeric()

Description

Returns true if the attribute data type handles numbers.

Syntax

isNumeric()

- Parameters

None

- Returns

True, if the attribute data type handles numbers.

Calculation Kind Types

The calculation kind type object contains the available types that SPC calculation can be:

Available data types:

- Anderson-Darling
- Control Limit
- Interval
- Parts Per Million
- Process Capability
- Process Capability & Process Performance
- Process Performance
- Signal Event

Properties:

intToType(ordinal)

Description

Returns the calculation kind type object for the ordinal value specified.

Syntax

intToType(ordinal)

- Parameters

int ordinal - Integer indicating position of the calculation kind object.

- Returns

CalculationKindTypes type - The calculation kind type object for the ordinal specified.

- Scope

All

getTypeFromDescription(description)

Description

Gets the ClaculationKindType object specified by the description parameter.

Syntax

getTypeFromDescription(description)

- Parameters

String description - Description of the calculation kind type to get the type for.

- Returns

Calculation KindTypes type - Type of calculation kind specified by the description parameter.

- Scope

All

getCategory()

Description

Returns the category of chart. See SPC Category Types for more information.

Syntax

getCategory()

- Parameters

None

- Returns

SPCCategoryTypes - The category object for the calculation kind is returned.

- Scope

All

isUserSelectable()

Description

Returns true if the event kind can be selected by the user.

Syntax

isUserSelectable()

- Parameters

None

- Returns

boolean - True if the event kind types can be selected by the user, False otherwise.

- Scope

All

getTypeFromName(name)

Description

Returns the calculation kind type object for the name value specified.

Syntax

getTypeFromName(name)

- Parameters

None

- Returns

Calculation KindTypes - The category object for the event kind is returned.

- Scope

All

Control Limit Calculated Value

When using the calcControlLimitValue functions, the new calculated control limit value and any messages are returned in this object. Most control limits are a single value across all samples. The p and u chart control limits can have different values for each sample. In this case, the results are returned in a Dataset that is also in this object.

Methods:

getCalculatedValue()

Description

Returns the new single control limit value.

Syntax

getCalculatedValue()

- Parameters

None

- Returns

Double value - The calculated value of the control limit.

- Scope

All

getData()

Description

Returns multiple control limit value that vary for each sample.

Syntax

getData()

- Parameters

None

- Returns

AnalysisDataset - The dataset containing control limit calculated value.

- Scope

All

getMessage()

Description

Message of why the control limit cannot be calculated.

Syntax

getMessage()

- Parameters

None

- Returns

String message - An error message regarding the control limit calculation failure.

- Scope

All

hasMessage()

Description

Returns True if a message exists.

Syntax

hasMessage()

- Parameters

None

- Returns

Boolean - True if the message exists and False otherwise.

- Scope

All

Control Limit Event

The event is created to calculate a control limit value.

getControlLimitValue()

Description

Returns the value of the control limit.

Syntax

getControlLimitValue()

- Parameters

None

- Returns

Double controlLimitValue - The value of the control limit.

- Scope

All

setControlLimitValue(controlLimitValue)

Description

Sets the control limit value.

Syntax

setControlLimitValue(controlLimitValue)

- Parameters

Double controlLimitValue - The value of the control limit.

- Returns

Nothing

- Scope

All

getControlLimitName()

Description

Returns the name of the control limit.

Syntax

getControlLimitName()

- Parameters

None

- Returns

String controlLimitName - The name of the control limit.

- Scope

All

getAttributeName()

Description

Returns the attribute name within the definition to set the control limit for.

Syntax

getAttributeName()

- Parameters

None

- Returns

String attributeName - The attribute name of the control limit.

- Scope

All

getData()

Description

Returns the SPC data.

Syntax

getData()

- Parameters

None

- Returns

AnalysisDataset data - The SPC data.

- Scope

All

setValue(rowIndex, columnIndex, value)

Description

Sets the value in the SPC data at the specified rowIndex and columnIndex.

Syntax

setValue(rowIndex, columnIndex, value)

- Parameters

None

- Returns

AnalysisDataset data - The SPC data.

- Scope

All

getSampleSize()

Description

Returns the sample size.

Syntax

getSampleSize()

- Parameters

None

- Returns

Integer sampleSize - The sample size associated with this control limit.

- Scope

All

getCalcValues()

Description

Returns the value information of the control limit.

Syntax

getCalcValues()

- Parameters

None

- Returns

SPCCalcValueCollection calcValues - The calculated value of the control limit.

- Scope

All

Control Limit Kind Type

The control limit kind type object contains the available types of control limits. In all cases, the ending of the name specifies how it is used in control charts and automatic signal evaluation. An ending of _UCL is handled as upper control limit, for _LCL it is handled as lower control limit and _OTHER is a general control limit.

Available data types:

XBAR_UCL

XBAR_LCL

XBAR_OTHER

Used for the XBar control chart.

RANGE_UCL

RANGE_LCL

RANGE_OTHER

Used for the Range control chart.

STDDEV_UCL

STDDEV_LCL

STDDEV_OTHER

Used for the s (standard deviation) control chart.

INDV_UCL

INDV_LCL

INDV_OTHER

Used for the Individual control chart.

MEDIAN_UCL

MEDIAN_LCL

MEDIAN_OTHER

Used for the Median control chart.

P_UCL

P_LCL

P_OTHER

Used for the p control chart.

NP_UCL

NP_LCL

NP_OTHER

Used for the np control chart.

C_UCL

C_LCL

C_OTHER

Used for the c control chart.

U_UCL

U_LCL

U_OTHER

Used for the u control chart.

HISTOGRAM_UCL

HISTOGRAM_LCL

HISTOGRAM_OTHER

Used for the Histogram chart.

MOVING_RANGE_UCL

MOVING_RANGE_LCL

MOVING_RANGE_OTHER

Used for the MA (moving average) control chart.

Properties:

getCategory()

Description

Returns the category of chart. See SPC Category Types for more information.

Syntax

getCategory()

- Parameters

None

- Returns

SPCCategoryTypes - The category this SPC chart belongs to.

getText()

Description

Returns the user friendly localized text for the control limit kind.

Syntax

getText()

- Parameters

None

- Returns

String text - The localized text for the control limit kind.

getTypeFromName(name)

Description

Returns the control limit kind type object for the name value specified.

Syntax

getTypeFromName(name)

- Parameters

String name - The name of the type to return for.

- Returns

ControlLimitKindTypes - The control limit kind type specified by the name parameter.

intToType(ordinal)

Description

Returns the control limit kind type object for the ordinal value specified.

Syntax

intToType(ordinal)

- Parameters

int ordinal - Integer indicating the position of the control limit value object to return for.

- Returns

ControlLimitKindTypes - The control limit kind type object specified by the ordinal.

MiscCalcEvent

The miscellaneous calculation event object is fired whenever the custom calculations through the system.quality.sample.data.executeMiscCalculation script function is done. This MES object is created for the execution of the miscellaneous calculation. If calculations other than the built-in calculations, such as PPM, are needed, then they can be defined in the Misc. Calculation section in the MES production model. Custom calculations can also be done using the tag change scripts (using system.quality.sample.data.executeMiscCalculation) where this event will execute the calculations.

Properties:

getData()

Description

Gets resultant dataset after the miscellaneous calculation.

Syntax

getData()

- Parameters

None

- Returns

AnalysisDataset data - The results of the miscellaneous calculation.

getMeasurementCount()

Description

Gets the number of measurements associated with this SPC data.

Syntax

getMeasurementCount()

- Parameters

None

- Returns

int measurementCnt - The measurement counts corresponding to this SPC data.

getSettings()

Description

Returns the SPC settings associated with this calculation event. An instance of a SPCSettings object that defines the samples to perform the calculation.

Syntax

getSettings()

- Parameters

None

- Returns

SPCSettings spcSettings - The settings for miscellaneous calculation.

getValue(key)

Description

Returns the value from the resultant dataset corresponding to the key parameter.

Syntax

getValue(key)

- Parameters

String key - The key to return the value for.

- Returns

Object value - The value corresponding to the key specified.

setValue(key, value)

Description

Sets the value for the miscellaneous calculation of SPC data.

Syntax

setValue(key, value)

- Parameters

String key - The key to set the data for.

Object value - The value to set the data for.

- Returns

Nothing

Parts Per Million Event

The event is created to calculate Parts Per Million (PPM).

Properties:

getActualStandardDeviation()

Description

Gets the actual standard deviation.

Syntax

getActualStandardDeviation()

- Parameters

None

- Returns

Double actualStandardDeviation

- Scope

All

getData()

Description

Gets the SPC data.

Syntax

getData()

- Parameters

None

- Returns

AnalysisDataset data

- Scope

All

getDefectPercentage()

Description

Gets the defect percentage.

Syntax

getDefectPercentage()

- Parameters

None

- Returns

Double defectPercentage

- Scope

All

getEstimatedStandardDeviation()

Description

Gets the estimated standard deviation.

Syntax

getEstimatedStandardDeviation()

- Parameters

None

- Returns

Double estimatedStandardDeviation

- Scope

All

getLsl()

Description

Gets the lower specification limit.

Syntax

getLsl()

- Parameters

None

- Returns

Double lsl

- Scope

All

getMean()

Description

Gets the mean.

Syntax

getMean()

- Parameters

None

- Returns

Double mean

- Scope

All

getMeasuredPpm()

Description

Gets the measured PPM.

Syntax

getMeasuredPpm()

- Parameters

None

- Returns

Double measuredPpm

- Scope

All

getMeasurementCount()

Description

Gets the measurementCount.

Syntax

getMeasurementCount()

- Parameters

None

- Returns

int measurementCount

- Scope

All

getOverallEstimatedPpm()

Description

Gets the overall estimated PPM.

Syntax

getOverallEstimatedPpm()

- Parameters

None

- Returns

Double overallEstimatedPpm

- Scope

All

getOverallPpml()

Description

Gets the overall PPML.

Syntax

getOverallPpml()

- Parameters

None

- Returns

Double overallPpml

- Scope

All

getOverallPpmu()

Description

Gets the overall PPMU.

Syntax

getOverallPpmu()

- Parameters

None

- Returns

Double overallPpmu

- Scope

All

getTarget()

Description

Gets the target.

Syntax

getTarget()

- Parameters

None

- Returns

Double target

- Scope

All

getUsl()

Description

Gets the upper specification limit.

Syntax

getUsl()

- Parameters

None

- Returns

Double usl

- Scope

All

getWithinEstimatedPpm()

Description

Gets the within estimated PPM.

Syntax

getWithinEstimatedPpm()

- Parameters

None

- Returns

Double withinEstimatedPpm

- Scope

All

getWithinPpml()

Description

Gets the within PPML.

Syntax

getWithinPpml()

- Parameters

None

- Returns

Double withinPpml

- Scope

All

getWithinPpmu()

Description

Gets the within PPMU.

Syntax

getWithinPpmu()

- Parameters

None

- Returns

Double withinPpmu

- Scope

All

setActualStandardDeviation(actualStandardDeviation)

Description

Sets the actual standard deviation.

Syntax

setActualStandardDeviation(actualStandardDeviation)

- Parameters

Double actualStandardDeviation

- Returns

Nothing

- Scope

All

setDefectPercentage(defectPercentage)

Description

Sets the defect percentage.

Syntax

setDefectPercentage(defectPercentage)

- Parameters

Double defectPercentage

- Returns

Nothing

- Scope

All

setEstimatedStandardDeviation(estimatedStandardDeviation)

Description

Sets the estimated standard deviation.

Syntax

setEstimatedStandardDeviation(estimatedStandardDeviation)

- Parameters

Double estimatedStandardDeviation

- Returns

Nothing

- Scope

All

setMean(mean)

Description

Sets the mean.

Syntax

setMean(mean)

- Parameters

Double mean

- Returns

Nothing

- Scope

All

setMeasuredPpm(measuredPpm)

Description

Sets the measured PPM.

Syntax

setMeasuredPpm(measuredPpm)

- Parameters

Double measuredPpm

- Returns

Nothing

- Scope

All

setOverallEstimatedPpm(overallEstimatedPpm)

Description

Sets the overall estimated PPM.

Syntax

setOverallEstimatedPpm(overallEstimatedPpm)

- Parameters

Double overallEstimatedPpm

- Returns

Nothing

- Scope

All

setOverallPpml(overallPpml)

Description

Sets the overall PPML.

Syntax

setOverallPpml(overallPpml)

- Parameters

Double overallPpml

- Returns

Nothing

- Scope

All

setOverallPpmu(overallPpmu)

Description

Sets the overall PPMU.

Syntax

setOverallPpmu(overallPpmu)

- Parameters

Double overallPpmu

- Returns

Nothing

- Scope

All

setWithinEstimatedPpm(withinEstimatedPpm)

Description

Sets the within estimated PPM.

Syntax

setWithinEstimatedPpm(withinEstimatedPpm)

- Parameters

Double withinEstimatedPpm

- Returns

Nothing

- Scope

All

setWithinPpml(withinPpml)

Description

Sets the within PPML.

Syntax

setWithinPpml(withinPpml)

- Parameters

Double withinPpml

- Returns

Nothing

- Scope

All

setWithinPpmu(withinPpmu)

Description

Sets the within PPMU.

Syntax

setWithinPpmu(withinPpmu)

- Parameters

Double withinPpmu

- Returns

Nothing

- Scope

All

Process Capability and Process Performance Event

The event is created to calculate a process capability and a process performance.

Properties:

getCp()

Description

Gets the process capability.

Syntax

getCp()

- Parameters

None

- Returns

Double cp - The process capability.

- Scope

All

getCpk()

Description

Gets the process capability index.

Syntax

getCpk()

- Parameters

None

- Returns

Double cpk - The index of the process capability.

- Scope

All

getCpl()

Description

Gets the lower process capability index.

Syntax

getCpl()

- Parameters

None

- Returns

Double cpl - The lower process capability index.

- Scope

All

getCpLcl()

Description

Gets the lower control limit of the process capability.

Syntax

getCpLcl()

- Parameters

None

- Returns

Double cpLcl - The lower control limit of the process capability.

- Scope

All

getCpm()

Description

Gets the process capability index of the mean.

Syntax

getCpm()

- Parameters

None

- Returns

Double cpm - The process capability index of the mean.

- Scope

All

getCpStandardDeviation()

Description

Gets the estimated standard deviation of the process capability.

Syntax

getCpStandardDeviation()

- Parameters

None

- Returns

Double cpStadardDeviation - The estimated standard deviation of the process capability.

- Scope

All

getCpu()

Description

Gets the upper process capability index.

Syntax

getCpu()

- Parameters

None

- Returns

Double cpu - The upper process capability index.

- Scope

All

getCpUcl()

Description

Gets the upper control limit of the process capability.

Syntax

getCpUcl()

- Parameters

None

- Returns

Double cpUcl- The upper control limit of the process capability.

- Scope

All

getCr()

Description

Gets the reciprocal of the process capability.

Syntax

getCr()

- Parameters

None

- Returns

Double cr - The reciprocal of the process capability.

- Scope

All

getData()

Description

Gets the SPC data.

Syntax

getData()

- Parameters

None

- Returns

AnalysisDataset data - The SPC data.

- Scope

All

getLsl()

Description

Gets the lower specification limit.

Syntax

getLsl()

- Parameters

None

- Returns

Double lsl - The lower specification limit.

- Scope

All

getMean()

Description

Gets the mean.

Syntax

getMean()

- Parameters

None

- Returns

Double mean - The mean.

- Scope

All

getMeasurementCount()

Description

Gets the measurement count.

Syntax

getMeasurementCount()

- Parameters

None

- Returns

int measurementCnt - The measurement count.

- Scope

All

getPp()

Description

Gets the process performance.

Syntax

getPp()

- Parameters

None

- Returns

Double pp - The process performance.

- Scope

All

getPpk()

Description

Gets the process performance index.

Syntax

getPpk()

- Parameters

None

- Returns

Double ppk - The process performance index.

- Scope

All

getPpl()

Description

Gets the lower process performance index.

Syntax

getPpl()

- Parameters

None

- Returns

Double ppl - The lower process performance index.

- Scope

All

getPpLcl()

Description

Gets the lower control limit of the process performance.

Syntax

getPpLcl()

- Parameters

None

- Returns

Double ppLcl - The lower control limit of the process performance.

- Scope

All

getPpm()

Description

Gets the process performance index of the mean.

Syntax

getPpm()

- Parameters

None

- Returns

Double ppm - The process performance index of the mean.

- Scope

All

getPpStandardDeviation()

Description

Gets the actual standard deviation of the process performance.

Syntax

getPpStandardDeviation()

- Parameters

None

- Returns

Double ppStandardDeviation - The actual standard deviation of the process performance.

- Scope

All

getPpu()

Description

Gets the upper process performance index.

Syntax

getPpu()

- Parameters

None

- Returns

Double ppu - The upper process performance index.

- Scope

All

getPpUcl()

Description

Gets the upper control limit of the process performance.

Syntax

getPpUcl()

- Parameters

None

- Returns

Double ppUCL - The upper control limit of the process performance.

- Scope

All

getPr()

Description

Gets the reciprocal of the process performance.

Syntax

getPr()

- Parameters

None

- Returns

Double pr - The reciprocal of the process performance.

- Scope

All

getTarget()

Description

Gets the target.

Syntax

getTarget()

- Parameters

None

- Returns

Double target - The target.

- Scope

All

getUsl()

Description

Gets the upper specification limit.

Syntax

getUsl()

- Parameters

None

- Returns

Double usl - The upper specification limit.

- Scope

All

setCp(cp)

Description

Sets the process capability.

Syntax

setCp(cp)

- Parameters

Double cp - The process capability to set for.

- Returns

Nothing

- Scope

All

setCpk(cpk)

Description

Sets the process capability index.

Syntax

setCpk(cpk)

- Parameters

Double cpk - The process capability index to set for.

- Returns

Nothing

- Scope

All

setCpl(cpl)

Description

Sets the lower process capability index.

Syntax

setCpl(cpl)

- Parameters

Double cpl - The lower process capability index to set for.

- Returns

Nothing

- Scope

All

setCpLcl(lcl)

Description

Sets the lower control limit of the process capability.

Syntax

setCpLcl(lcl)

- Parameters

Double lcl - The lower control limit of the process capability to set for.

- Returns

Nothing

- Scope

All

setCpm(cpm)

Description

Sets the process capability index of the mean.

Syntax

setCpm(cpm)

- Parameters

Double cpm - The process capability index of the mean to set for.

- Returns

Nothing

- Scope

All

setCpStandardDeviation(standardDeviation)

Description

Sets the estimated standard deviation of the process capability.

Syntax

setCpStandardDeviation(standardDeviation)

- Parameters

Double standardDeviation - The estimated standard deviation of the process capability to set for.

- Returns

Nothing

- Scope

All

setCpu(cpu)

Description

Sets the upper process capability index.

Syntax

setCpu(cpu)

- Parameters

Double cpu - The upper process capability index to set for.

- Returns

Nothing

- Scope

All

setCpUcl(ucl)

Description

Sets the upper control limit of the process capability.

Syntax

setCpUcl(ucl)

- Parameters

Double ucl - The upper control limit of the process capability to set for.

- Returns

Nothing

- Scope

All

setCr(cr)

Description

Sets the reciprocal of the process capability.

Syntax

setCr(cr)

- Parameters

Double cr - The reciprocal of the process capability to set for.

- Returns

Nothing

- Scope

All

setMean(mean)

Description

Sets the mean.

Syntax

setMean(mean)

- Parameters

Double mean - The mean to set for.

- Returns

Nothing

- Scope

All

setPp(pp)

Description

Sets the process performance.

Syntax

setPp(pp)

- Parameters

Double pp - The process performance to set for.

- Returns

Nothing

- Scope

All

setPpk(ppk)

Description

Sets the process performance index.

Syntax

setPpk(ppk)

- Parameters

Double ppk - The process performance index to set for.

- Returns

Nothing

- Scope

All

setPpl(ppl)

Description

Sets the lower process performance index.

Syntax

setPpl(ppl)

- Parameters

Double ppl - The lower process performance index to set for.

- Returns

Nothing

- Scope

All

setPpLcl(lcl)

Description

Sets the lower control limit of the process performance.

Syntax

setPpLcl(lcl)

- Parameters

Double lcl - The lower control limit of the process performance to set for.

- Returns

Nothing

- Scope

All

setPpm(ppm)

Description

Sets the process performance index of the mean.

Syntax

setPpm(ppm)

- Parameters

Double ppm - The process performance index of the mean to set for.

- Returns

Nothing

- Scope

All

setPpStandardDeviation(standardDeviation)

Description

Sets the actual standard deviation of the process performance.

Syntax

setPpStandardDeviation(standardDeviation)

- Parameters

Double standardDeviation - The actual standard deviation of the process performance.

- Returns

Nothing

- Scope

All

setPpu(ppu)

Description

Sets the upper process performance index.

Syntax

setPpu(ppu)

- Parameters

Double ppu - The upper process performance index to set for.

- Returns

Nothing

- Scope

All

setPpUcl(ucl)

Description

Sets the upper control limit of the process performance.

Syntax

setPpUcl(ucl)

- Parameters

Double ucl - The upper control limit of the process performance to set for.

- Returns

Nothing

- Scope

All

setPr(pr)

Description

Sets the reciprocal of the process performance.

Syntax

setPr(pr)

- Parameters

Double pr - The reciprocal of the process performance to set for.

- Returns

Nothing

- Scope

All

Process Capability Event

The event is created to calculate a process capability.

Properties:

getCp()

Description

Gets the process capability.

Syntax

getCp()

- Parameters

None

- Returns

Double cp - The process capability.

- Scope

All

getCpk()

Description

Gets the process capability index.

Syntax

getCpk()

- Parameters

None

- Returns

Double cpk - The index of process capability.

- Scope

All

getCpl()

Description

Gets the lower process capability index.

Syntax

getCpl()

- Parameters

None

- Returns

Double cpl - The lower process capability index.

- Scope

All

getCpm()

Description

Gets the process capability mean.

Syntax

getCpm()

- Parameters

None

- Returns

Double cpm - The process capability mean.

- Scope

All

getCpu()

Description

Gets the upper process capability index.

Syntax

getCpu()

- Parameters

None

- Returns

Double cpu - The upper process capability index.

- Scope

All

getCr()

Description

Gets the reciprocal of process capability.

Syntax

getCr()

- Parameters

None

- Returns

Double cr - The reciprocal of process capability.

- Scope

All

getData()

Description

Gets the SPC data.

Syntax

getData()

- Parameters

None

- Returns

AnalysisDataset data - The SPC data.

- Scope

All

getLcl()

Description

Gets the lower control limit.

Syntax

getLcl()

- Parameters

None

- Returns

Double lcl - The lower control limit.

- Scope

All

getLsl()

Description

Gets the lower specification limit.

Syntax

getLsl()

- Parameters

None

- Returns

Double lsl - The lower specification limit.

- Scope

All

getMean()

Description

Gets the mean.

Syntax

getMean()

- Parameters

None

- Returns

Double mean - The mean.

- Scope

All

getMeasurementCount()

Description

Gets the measurement count.

Syntax

getMeasurementCount()

- Parameters

None

- Returns

int measurementCnt - The measurement count.

- Scope

All

getStandardDeviation()

Description

Gets the estimated standard deviation.

Syntax

getStandardDeviation()

- Parameters

None

- Returns

Double standardDeviation - The estimated standard deviation.

- Scope

All

getTarget()

Description

Gets the target.

Syntax

getTarget()

- Parameters

None

- Returns

Double target - The target.

- Scope

All

getUcl()

Description

Gets the upper control limit.

Syntax

getUcl()

- Parameters

None

- Returns

Double ucl - The upper control limit.

- Scope

All

getUsl()

Description

Gets the upper specification limit.

Syntax

getUsl()

- Parameters

None

- Returns

Double usl - The upper specification limit.

- Scope

All

setCp(cp)

Description

Sets the process capability.

Syntax

setCp(cp)

- Parameters

Double cp - The process capability to set for.

- Returns

Nothing

- Scope

All

setCpk(cpk)

Description

Sets the process capability index.

Syntax

setCpk(cpk)

- Parameters

Double cpk - The process capability index to set for.

- Returns

Nothing

- Scope

All

setCpl(cpl)

Description

Sets the lower process capability index.

Syntax

setCpl(cpl)

- Parameters

Double cpl - The lower process capability index to set for.

- Returns

Nothing

- Scope

All

setCpm(cpm)

Description

Sets the process capability mean.

Syntax

setCpm(cpm)

- Parameters

Double cpm - The process capability mean to set for.

- Returns

Nothing

- Scope

All

setCpu(cpu)

Description

Sets the upper process capability index.

Syntax

setCpu(cpu)

- Parameters

Double cpu - The upper process capability index to set for.

- Returns

Nothing

- Scope

All

setCr(cr)

Description

Sets the reciprocal of process capability.

Syntax

setCr(cr)

- Parameters

Double cr - The reciprocal of process capability.

- Returns

Nothing

- Scope

All

setLcl(lcl)

Description

Sets the lower control limit.

Syntax

setLcl(lcl)

- Parameters

Double lcl - The lower control limit to set for.

- Returns

Nothing

- Scope

All

setMean(mean)

Description

Sets the mean.

Syntax

setMean(mean)

- Parameters

Double mean - The mean to set for.

- Returns

Nothing

- Scope

All

setStandardDeviation(standardDeviation)

Description

Sets the estimated standard deviation.

Syntax

setStandardDeviation(standardDeviation)

- Parameters

Double standardDeviation - The estimated standard deviation to set for.

- Returns

Nothing

- Scope

All

setUcl(ucl)

Description

Sets the upper control limit.

Syntax

setUcl(ucl)

- Parameters

Double ucl - The upper control limit to set for.

- Returns

Nothing

- Scope

All

Process Performance Event

The event is created to calculate a process performance.

Properties:

getData()

Description

Gets the SPC data.

Syntax

getData()

- Parameters

None

- Returns

AnalysisDataset data - The SPC data.

- Scope

All

getLcl()

Description

Gets the lower control limit.

Syntax

getLcl()

- Parameters

None

- Returns

Double lcl - The lower control limit.

- Scope

All

getLsl()

Description

Gets the lower specification limit.

Syntax

getLsl()

- Parameters

None

- Returns

Double lsl - The lower specification limit.

- Scope

All

getMean()

Description

Gets the mean.

Syntax

getMean()

- Parameters

None

- Returns

Double mean - The mean.

- Scope

All

getMeasurementCount()

Description

Gets the measurement count.

Syntax

getMeasurementCount()

- Parameters

None

- Returns

int measurementCnt - The measurement count.

- Scope

All

getPp()

Description

Gets the process performance.

Syntax

getPp()

- Parameters

None

- Returns

Double pp - The process performance.

- Scope

All

getPpk()

Description

Gets the process performance index.

Syntax

getPpk()

- Parameters

None

- Returns

Double ppk - The process performance index.

- Scope

All

getPpl()

Description

Gets the lower process performance index.

Syntax

getPpl()

- Parameters

None

- Returns

Double ppl - The lower process performance index.

- Scope

All

getPpm()

Description

Gets the process performance index of the mean.

Syntax

getPpm()

- Parameters

None

- Returns

Double ppm - The process performance index of the mean.

- Scope

All

getPpu()

Description

Gets the upper process performance index.

Syntax

getPpu()

- Parameters

None

- Returns

Double ppu - The upper process performance index.

- Scope

All

getPr()

Description

Gets the reciprocal of the process performance.

Syntax

getPr()

- Parameters

None

- Returns

Double pr - The reciprocal of the process performance.

- Scope

All

getStandardDeviation()

Description

Gets the actual standard deviation.

Syntax

getStandardDeviation()

- Parameters

None

- Returns

Double standardDeviation - The actual standard deviation.

- Scope

All

getTarget()

Description

Gets the target.

Syntax

getTarget()

- Parameters

None

- Returns

Double target - The target.

- Scope

All

getUcl()

Description

Gets the upper control limit.

Syntax

getUcl()

- Parameters

None

- Returns

Double ucl - The upper control limit.

- Scope

All

getUsl()

Description

Gets the upper specification limit.

Syntax

getUsl()

- Parameters

None

- Returns

Double usl - The upper specification limit.

- Scope

All

setLcl(lcl)

Description

Sets the lower control limit.

Syntax

setLcl(lcl)

- Parameters

Double lcl - The lower specification limit to set for.

- Returns

Nothing

- Scope

All

setMean(mean)

Description

Sets the mean.

Syntax

setMean(mean)

- Parameters

Double mean - The mean to set for.

- Returns

Nothing

- Scope

All

setPp(pp)

Description

Sets the process performance.

Syntax

setPp(pp)

- Parameters

Double pp - The process performance to set for.

- Returns

Nothing

- Scope

All

setPpk(ppk)

Description

Sets the process performance index.

Syntax

setPpk(ppk)

- Parameters

Double ppk - The process performance index to set for.

- Returns

Nothing

- Scope

All

setPpl(ppl)

Description

Sets the lower process performance index.

Syntax

setPpl(ppl)

- Parameters

Double ppl - The lower process performance index to set for.

- Returns

Nothing

- Scope

All

setPpm(ppm)

Description

Sets the process performance index of the mean.

Syntax

setPpm(ppm)

- Parameters

Double ppm - The process performance index of the mean to set for.

- Returns

Nothing

- Scope

All

setPpu(ppu)

Description

Sets the upper process performance index.

Syntax

setPpu(ppu)

- Parameters

Double ppu - The upper process performance index to set for.

- Returns

Nothing

- Scope

All

setPr(pr)

Description

Sets the reciprocal of the process performance.

Syntax

setPr(pr)

- Parameters

Double pr - The reciprocal of the process performance to set for.

- Returns

Nothing

- Scope

All

setStandardDeviation(standardDeviation)

Description

Sets the actual standard deviation.

Syntax

setStandardDeviation(standardDeviation)

- Parameters

Double standardDeviation - The actual standard deviation to set for.

- Returns

Nothing

- Scope

All

setUcl(ucl)

Description

Sets the upper control limit.

Syntax

setUcl(ucl)

- Parameters

Double ucl - The upper control limit to set for.

- Returns

Nothing

- Scope

All

Sample

The sample object holds all of the information associated with one sample.

Properties:

calcScheduledFinish()

Description

Based on the scheduled start date and time and the duration of time to take this sample, calculates the date and time this sample is scheduled to be complete. The getScheduledFinish() value is updated after calling this function. The duration of time required to take a sample is defined in the sample definition. For automatic samplings, this value does not apply.

Syntax

calcScheduledFinish()

- Parameters

None

- Returns

Nothing

getApproved()

Description

Returns true if this sample has been approved. Depending on the settings in the sample definition, samples may be automatically or manually approved.

Syntax

getApproved()

- Parameters

None

- Returns

True, if the sample has been approved.

getApprovedBy()

Description

Returns the person’s name who approved this sample. For automatically recorded samples, this will be “Auto”.

Syntax

getApprovedBy()

- Parameters

None

- Returns

String approvedBy - Name of the person who approved this sample.

getApprovedDateTime()

Description

Returns the date and time that this sample was approved. For automatically approved samples, this will be the same as the getEntryDateTime() value.

Syntax

getApprovedDateTime()

- Parameters

None

- Returns

Calendar approvedDateTime - Date and time that this sample was approved.

getArea()

Description

Returns the production area associated with this sample.

Syntax

getArea()

- Parameters

None

- Returns

String area - The area associated with the sample.

getDefUUID()

Description

Returns the definition UUID associated with this sample. (See Sample Definition object for more information).

Syntax

getDefUUID()

- Parameters

None

- Returns

String defUUID - The definition uuid of this sample.

getEnterprise()

Description

Returns the enterprise associated with this sample.

Syntax

getEnterprise()

- Parameters

None

- Returns

String enterprise - The enterprise of this sample.

getEntryDateTime()

Description

Returns the date and time that this sample was entered.

Syntax

getEntryDateTime()

- Parameters

None

- Returns

Calendar entryDateTime - Date and time for which the sample was entered.

getId()

Description

Returns the id associated with this sample.

Syntax

getId()

- Parameters

None

- Returns

Integer id - The id of this sample.

getLine()

Description

Returns the production line associated with this sample. This will be blank if the location the sample is taken is in a production area and not on a production line.

Syntax

getLine()

- Parameters

None

- Returns

String line - The line associated with the sample.

getLocation()

Description

Returns the location associated with this sample.

Syntax

getLocation()

- Parameters

None

- Returns

String location - The location associated with this sample.

getLocationPath()

Description

Returns the full location path, including enterprise, site, area, line and location, associated with this sample.

Syntax

getLocationPath()

- Parameters

None

- Returns

String locationPath - The location path of this sample.

getNote()

Description

Returns the note associated with this sample. This is the note that may have been entered when the sample was entered. Even though this note can be viewed on the control charts or in analysis, it is not the same as the attribute note entered on the control charts.

Syntax

getNote()

- Parameters

None

- Returns

String note - Note associated with the sample.

getProductCode()

Description

Returns the product code associated with this sample. This is optional and may not apply if tracking quality by product code is not being used for the associated sample definition.

Syntax

getProductCode()

- Parameters

None

- Returns

String productCode - The product code associated with the sample.

getRefNo()

Description

Returns the reference number associated with this sample. This is optional and can be used to track information like batch number, lot number, etc. Additional factors can also be used to track information.

Syntax

getRefNo()

- Parameters

None

- Returns

String refNo - The reference number associated with the sample.

getSampleDefinition()

Description

Returns the definition associated with this sample. See Sample Definition object for more information.

Syntax

getSampleDefinition()

- Parameters

None

- Returns

SampleDefinition definition - Definition associated with the sample.

getSampleTakenBy()

Description

Returns the person's name who was responsible for taking the sample. By default, this is the person who is logged in when the sample is entered. For automatically recorded samples, this will be "Auto".

Syntax

getSampleTakenBy()

- Parameters

None

- Returns

String sampleTakenBy - Name of person who took the sample.

getSampleTakenDateTime()

Description

Returns the date and time that this sample was taken.

Syntax

getSampleTakenDateTime()

- Parameters

None

- Returns

Calendar sampleTakenDateTime - The date and time that the sample was taken.

getSampleUUID()

Description

Returns the UUID assigned to this sample. A UUID is a universally unique identifier that, once assigned to a sample, will never change. It is also unique in that no two samples will have the same UUID.

Syntax

getSampleUUID()

- Parameters

None

- Returns

String sampleUUID - The uuid of this sample.

getScheduledFinish()

Description

Returns the date and time that taking this sample is scheduled to be complete. For automatic samplings, this value does not apply and will be equal to None.

Syntax

getScheduledFinish()

- Parameters

None

- Returns

Calendar scheduledFinish - The date and time for the schedule to end.

getScheduledStart()

Description

Returns the date and time that this sample is scheduled to be taken. For automatic samplings, this value does not apply and will be equal to None.

Syntax

getScheduledStart()

- Parameters

None

- Returns

Calendar scheduledStart - The date and time for the sample to start.

getSequenceDate()

Description

Returns the date and time that the shift the sample was taken during started.

Syntax

getSequenceDate()

- Parameters

None

- Returns

Calendar sequenceDate - Start date of the current active shift.

getShift()

Description

Returns the shift the sample was taken.

Syntax

getShift()

- Parameters

None

- Returns

int shift - Shift for which the sample was taken.

getSite()

Description

Returns the physical production facility associated with this sample.

Syntax

getSite()

- Parameters

None

- Returns

String site - The site of this sample.

getTag()

Description

Returns the optional tag value. This is typically used to assign ownership of which department has the responsibility to take this sample.

Syntax

getTag()

- Parameters

None

- Returns

String tag - Tag value that specifies the ownership.

isModified()

Description

Returns true if any properties of this sample have been modified.

Syntax

isModified()

- Parameters

None

- Returns

True, if this sample have been modified.

isNew()

Description

Returns true if this is a newly created sample.

Syntax

isNew()

- Parameters

None

- Returns

True, if this sample is newly created.

setApproved(approved)

Description

Set to true to approve this sample.

Syntax

setApproved(approved)

- Parameters

boolean approved - True, if the sample is to be approved.

- Returns

Nothing

setApprovedBy(approvedBy)

Description

Sets the person’s name who approved this sample.

Syntax

setApprovedBy(approvedBy)

- Parameters

String approvedBy - Name of the person who approved this sample.

- Returns

Nothing

setApprovedDateTime(approvedDateTime)

Description

Sets the date and time that this sample was approved.

Syntax

setApprovedDateTime(approvedDateTime)

- Parameters

Calendar approvedDateTime - Date and time that this sample was approved.

- Returns

Nothing

setArea(area)

Description

Sets the production area associated with this sample.

Syntax

setArea(area)

- Parameters

String area - The area associated with the sample.

- Returns

Nothing

setEnterprise(enterprise)

Description

Sets the enterprise associated with this sample.

Syntax

setEnterprise(enterprise)

- Parameters

String enterprise - The enterprise of this sample.

- Returns

Nothing

setEntryDateTime(entryDateTime)

Description

Sets the date and time that this sample was entered.

Syntax

setEntryDateTime(entryDateTime)

- Parameters

Calendar entryDateTime - Date and time for which the sample was entered.

- Returns

Nothing

setLine(line)

Description

Sets the production line associated with this sample.

Syntax

setLine(line)

- Parameters

String line - The line associated with the sample.

- Returns

Nothing

setLocation(location)

Description

Sets the production location associated with this sample.

Syntax

setLocation(location)

- Parameters

String location - The location associated with the sample.

- Returns

Nothing

setLocationPath(locationPath)

Description

Sets the enterprise, site, area, line and location from the locationPath parameter.

Syntax

setLocationPath(locationPath)

- Parameters

String locationPath - The locationPath of this sample.

- Returns

Nothing

setNote(note)

Description

Sets the note associated with this sample.

Syntax

setNote(note)

- Parameters

String note - Note associated with the sample.

- Returns

Nothing

setProductCode(productCode)

Description

Sets the product code associated with this sample.

Syntax

setProductCode(productCode)

- Parameters

String productCode - The product code associated with the sample.

- Returns

Nothing

setRefNo(refNo)

Description

Sets the reference number associated with this sample.

Syntax

setRefNo(refNo)

- Parameters

String refNo - The reference number of this sample.

- Returns

Nothing

setSampleDefinition(definition)

Description

Sets the definition associated with this sample. See Sample Definition object for more information.

Syntax

setSampleDefinition(definition)

- Parameters

SampleDefinition definition - Definition associated with the sample.

- Returns

Nothing

setSampleTakenBy(sampleTakenBy)

Description

Sets the person's name who was responsible for taking the sample.

Syntax

setSampleTakenBy(sampleTakenBy)

- Parameters

String sampleTakenBy - Name of person who took the sample.

- Returns

Nothing

setSampleTakenDateTime(sampleTakenDateTime)

Description

Sets the date and time that this sample was taken.

Syntax

setSampleTakenDateTime(sampleTakenDateTime)

- Parameters

Calendar sampleTakenDateTime - Date and time that the sample was taken.

- Returns

Nothing

setScheduledFinished(scheduleFinish)

Description

Sets the date and time that this sample is scheduled to be completed.

Syntax

setScheduledFinished(scheduleFinish)

- Parameters

Calendar scheduleFinish - The date and time for the schedule to end.

- Returns

Nothing

setScheduledStart(scheduleStart)

Description

Sets the date and time that this sample is scheduled to be taken.

Syntax

setScheduledStart(scheduleStart)

- Parameters

Calendar scheduleStart - The date and time for the schedule to start.

- Returns

Nothing

setSequenceDate(sequenceDate)

Description

Sets the date and time that the shift the sample was taken during started.

Syntax

setSequenceDate(sequenceDate)

- Parameters

Calendar sequenceDate - Start date of the current active shift.

- Returns

Nothing

setShift(shift)

Description

Sets the shift the sample was taken.

Syntax

setShift(shift)

- Parameters

int shift - The shift for which the sample was taken.

- Returns

Nothing

setSite(site)

Description

Sets the physical production site associated with this sample.

Syntax

setSite(site)

- Parameters

String site - The site of this sample.

- Returns

Nothing

setTag(tag)

Description

Sets the tag value. This is typically used to assign ownership of which department has the responsibility to take this sample.

Syntax

setTag(tag)

- Parameters

String tag - Tag value that specifies the ownership.

- Returns

Nothing

Attribute properties:

getAttributeDataType(attrName)

Description

Returns the attribute data type object for the specified attribute name. The attribute data type information is contained in the sample definition and cannot be changed directly in the sample object.

Syntax

getAttributeDataType(attrName)

- Parameters

String attrName - Name of the attribute to return the data type for.

- Returns

AttributeDataType datatype - Attribute data type object for the specified attribute name.

getAttributeDefaultValue(attrName)

Description

Returns the default value for the specified attribute name. The attribute default value is contained in the sample definition and cannot be changed directly in the sample object.

Syntax

getAttributeDefaultValue(attrName)

- Parameters

String attrName - Name of the attribute to return the default value for.

- Returns

Object defaultValue - Default value for the attribute.

getAttributeMaxValue(attrName)

Description

Returns the maximum value for the specified attribute name. The attribute maximum value is contained in the sample definition and cannot be changed directly in the sample object.

Syntax

getAttributeMaxValue(attrName)

- Parameters

String attrName - Name of the attribute to return the maximum value for.

- Returns

Object maxValue - Maximum value for the attribute.

getAttributeMinValue(attrName)

Description

Returns the minimum value for the specified attribute name. The attribute minimum value is contained in the sample definition and cannot be changed directly in the sample object.

Syntax

getAttributeMinValue(attrName)

- Parameters

String attrName - Name of the attribute to return the minimum value for.

- Returns

Object minValue - Minimum value for the attribute.

Measurement properties:

getAllMeasurements()

Description

Returns the measurements associated with this sample. If a sample has been scheduled but the measurement data has not been recorded, the measurement entries will still exist. In this case, use the sampleDataExists() property to determine if the measurement data has been entered.

Syntax

getAllMeasurements()

- Parameters

None

- Returns

List - A list of all the measurements associated with this sample.

getSampleData(measNo, attrName)

Description

Gets SampleData item for the specified measurement number and attribute.

Syntax

getSampleData(measNo, attrName)

- Parameters

int measNo - The measurement number associated with the sample.

String attrName - Name of the attribute to return the maximum value for.

- Returns

SampleData data - SampleData item for the specified measurement number and attribute.

getSampleDataValue(measNo, attrName)

Description

Returns a measurement value as a string for the specified measurement number and attribute name.

Syntax

getSampleDataValue(measNo, attrName)

- Parameters

int measNo - The measurement number associated with the sample.

String attrName - Name of the attribute to return the measurement value for.

- Returns

String value - Measurement value for the specified sample.

isDataModified()

Description

Returns true if any measurement values have been modified.

Syntax

isDataModified()

- Parameters

None

- Returns

True, if any of the measurement value was modified.

isSampleDataValid()

Description

Returns true if the measurements have been entered and are valid.

Syntax

isSampleDataValid()

- Parameters

None

- Returns

True, if the measurements have been entered and are valid.

sampleDataExists(sampleData)

Description

Returns true if the given sample data exist and False otherwise.

Syntax

sampleDataExists(sampleData)

- Parameters

SampleData sampleData - The sample data to be checked.

- Returns

True, if sample data exists and False otherwise.

setSampleData(measNo, attrName, value)

Description

Sets a measurement value as a string for the specified measurement number and attribute name.

Syntax

setSampleData(measNo, attrName, value)

- Parameters

int measNo - The measurement number associated with the sample.

String attrName - Name of the attribute to set the sample data for.

String value - Measurement value for the specified sample.

- Returns

True if successful, otherwise returns false.

additional factor properties:

getAddlFactor(factorName)

Description

Gets the additional factor object specified by the factorName parameter and associated with this sample. Use this function to get the SampleAdditionalFactor object that can be used to change the value of the additional factor.

Syntax

getAddlFactor(factorName)

- Parameters

String factorName - Name of the additional factor to be returned. This reflects the name of the additional factor that is configured in the designer.

- Returns

SampleAdditionalFactor - A ddditional factor object associated with this sample.

getAllAddlFactors()

Description

Returns the list of additional factor values associated with this sample.

Syntax

getAllAddlFactors()

- Parameters

None

- Returns

List - A list of additional factor values associated with this sample.

Sample Additional Factor

The sample additional factor object holds all of the information associated with one sample.

Properties:

getDataType()

Description

Returns the data type of this additional factor. See DataType in the Ignition documentation for more information.

Syntax

getDataType()

- Parameters

None

- Returns

The data type of this additional factor.

getName()

Description

Returns the name of this additional factor.

Syntax

getName()

- Parameters

None

- Returns

Name of the additional factor.

getRecordDateTime()

Description

Returns the date and time the value was recorded for this additional factor.

Syntax

getRecordDateTime()

- Parameters

None

- Returns

Date and time the value was recorded for this additional factor.

getValue()

Description

Returns the value for this additional factor.

Syntax

getValue()

- Parameters

None

- Returns

Object value - The value of this additional factor.

isModified()

Description

Returns true if this additional factor has been modified.

Syntax

isModified()

- Parameters

None

- Returns

True, if this additional factor has been modified.

isNew()

Description

Returns true if this additional factor has been modified.

Syntax

isNew()

- Parameters

None

- Returns

True, if this additional factor has been modified.

resetModified()

Description

This script function will undo the modifications.

Syntax

resetModified()

- Parameters

None

- Returns

Nothing

updateValue(value, recordDateTime)

Description

Updates the value and the required date and time that is being recorded for this additional factor.

Syntax

updateValue(value, recordDateTime)

- Parameters

Object value - The value to be updated.

Date recordDateTime - The date and time that this value is recorded.

- Returns

Nothing

Sample Data

The sample object holds a sample data object for each attribute and measurement. When a sample object is created, it automatically creates a sample data object based on the sample definition.

Example:

If sample definition viscosity has two attributes of cold viscosity and temperature and is configured for 5 measurements, then the sample will contain 10 sample data objects. Five measurements for cold viscosity and five measurements for temperature.

Properties:

getAttrDataType()

Description

Returns attribute data type of this sample data object. This is automatically set when the sample is created and is based on the sample definition.

Syntax

getAttrDataType()

- Parameters

None

- Returns

AttributeDataType The attribute data type of this sample data object.

getAttrName()

Description

Returns the attribute name this sample data object is associated with.

Syntax

getAttrName()

- Parameters

None

- Returns

String attrName - The attribute name of this sample data.

getAttrValue()

Description

Returns the data value for this sample data object.

Syntax

getAttrValue()

- Parameters

None

- Returns

Object attrValue - The data value for this sample data.

getAttrValueAsString()

Description

Returns the value for this sample data object as a string.

Syntax

getAttrValueAsString()

- Parameters

None

- Returns

The attribute value as string.

getDefaultValue()

Description

Returns the default value based on the attribute this sample data object is associated with. This is automatically set when the sample is created and is based on the sample definition.

Syntax

getDefaultValue()

- Parameters

None

- Returns

Object value - The default value based on the attribute of this sample data.

getId()

Description

Gets the id for this sample data.

Syntax

getId()

- Parameters

None

- Returns

Integer id - The identifier of this sample.

getMaxValue()

Description

Returns the maximum value based on the attribute this sample data object is associated with. This is automatically set when the sample is created and is based on the sample definition.

Syntax

getMaxValue()

- Parameters

None

- Returns

Object value - The maximum value for this sample data.

getMeasNo()

Description

Returns the measurement number this sample data object is associated with.

Syntax

getMeasNo()

- Parameters

None

- Returns

Integer measNo - The measurement number of this sample data.

getMeasured()

Description

Boolean indicating whether this sample data is measured or not.

Syntax

getMeasured()

- Parameters

None

- Returns

boolean isMeasured - True, if this sample data is measured and False otherwise.

getMinValue()

Description

Returns the minimum value based on the attribute this sample data object is associated with. This is automatically set when the sample is created and is based on the sample definition.

Syntax

getMinValue()

- Parameters

None

- Returns

Object value - The minimum value for this sample data.

getSampleUUID()

Description

Returns the sample UUID that this sample data object belongs to.

Syntax

getSampleUUID()

- Parameters

None

- Returns

String sampleUUID - The uuid of this sample.

getSampleValidationErrors()

Description

Validates the sample and returns an error message if the sample is invalid.

Syntax

getSampleValidationErrors()

- Parameters

None

- Returns

String result - The message explaining why the sample is invalid.

hasAttrValue()

Description

Returns true if attribute value is not equal to null.

Syntax

hasAttrValue()

- Parameters

None

- Returns

boolean - True if there is an attribute value associated with this sample data.

isModified()

Description

Returns true if the value of this sample data object has been modified.

Syntax

isModified()

- Parameters

None

- Returns

True, if value of the sample has been modified.

isNew()

Description

Returns true if the value of this sample data object is a newly created one.

Syntax

isNew()

- Parameters

None

- Returns

True, if value of the sample has unassigned ID.

isValueValid()

Description

Returns true if the value of this sample data object has been set and is between minimum and maximum values.

Syntax

isValueValid()

- Parameters

None

- Returns

True, if value of the sample is valid.

resetModified()

Description

This script function will undo the modification.

Syntax

resetModified()

- Parameters

None

- Returns

Nothing

setAttrValue(attrValue)

Description

Sets the value for this sample data object. If the attrValue parameter is not the correct data type, an attempt to convert it to the correct data type is performed before it is set.

Syntax

setAttrValue(attrValue)

- Parameters

Object attrValue - The data value to set the sample data for.

- Returns

Nothing

setMeasured(isMeasured)

Description

Sets the boolean indicating that the sample data is measured or not.

Syntax

setMeasured(isMeasured)

- Parameters

boolean isMeasured - Set to True if the sample data is already measured or else set to False.

- Returns

Nothing

toString()

Description

Gets the attribute value as string.

Syntax

toString()

- Parameters

None

- Returns

The attribute value as string.

Sample Definition

The sample definition object holds all of the information defining a type of sample. A sample definition specifies the attributes to collect, the locations where sample data is collected from, the control limits that apply, and the signals that apply.

When samples are created, they are based on a sample definition. When sample measurement values are recorded, the sample definition is used to validate the measurement values. Other operations also refer back to the sample definition such as automatic scheduling of samples, auto evaluation of signals, etc.

Properties:

getAutoApprove()

Description

Returns the default auto approve samples setting for this sample definition. Allowed locations that belong to this sample definition are initialized with this default setting.

Syntax

getAutoApprove()

- Parameters

None

- Returns

Boolean autoApprove - True, if the default value of the auto approve setting for this sample definition is true and False otherwise.

getComingDueMin()

Description

Returns the default coming due minutes setting for this sample definition. Allowed locations that belong to this sample definition are initialized with this default setting. The value represents the number of minutes required before a sample is due until the sample is considered coming due. For automatically scheduled samples, they are created prior to actual due time by the number of minutes of this setting.

Syntax

getComingDueMin()

- Parameters

None

- Returns

double comingDueMin - The number of minutes required before a sample is due until it is coming due.

getDefUUID()

Description

Returns the UUID assigned to this sample definition. A UUID is a universally unique identifier that, once assigned to a sample definition, will never change. It is automatically generated when a sample definition is created and is unique in that no two samples definitions will have the same UUID.

Syntax

getDefUUID()

- Parameters

None

- Returns

String defUUID - The unique identifier allotted to this sample definition.

getDescription()

Description

Returns the description of this sample definition.

Syntax

getDescription()

- Parameters

None

- Returns

String description - The description given to this sample definition.

getEnabled()

Description

Returns true if sample definition is enabled.

Syntax

getEnabled()

- Parameters

None

- Returns

Boolean True, if the sample definition is enabled and False otherwise.

getId()

Description

Returns ID of the sample definition.

Syntax

getId()

- Parameters

None

- Returns

The ID of the sample definition.

getInterval()

Description

Returns the default interval for automatically scheduled samples based on this sample definition. Allowed locations that belong to this sample definition are initialized with this default interval. The units are defined by the Interval type defined for this sample definition.

Syntax

getInterval()

- Parameters

None

- Returns

Double interval - The interval associated with this scheduled samples.

getMeasurementCount()

Description

Returns the number of measurements that this sample definition is configured for.

Syntax

getMeasurementCount()

- Parameters

None

- Returns

Integer measurementCount - The number of measurements associated with this sample definition.

getName()

Description

Gets the name of this sample definition.

Syntax

getName()

- Parameters

None

- Returns

String name - The name of the sample definition.

getOverdueMin()

Description

Returns the default overdue minutes setting for this sample definition. Allowed locations that belong to this sample definition are initialized with this default setting.

Syntax

getOverdueMin()

- Parameters

None

- Returns

double overdueMin - The value represents the number of minutes required after a sample is due until the sample is considered overdue.

isModified()

Description

Returns true if this sample definition has been modified.

Syntax

isModified()

- Parameters

None

- Returns

True, if the sample definition is modified.

isNew()

Description

Returns true if this sample definition is new.

Syntax

isNew()

- Parameters

None

- Returns

True if the specified sample definition is new.

setAutoApprove(autoApprove)

Description

Sets the default auto approve samples setting for this sample definition. Allowed locations that belong to this sample definition are initialized with this default setting.

Syntax

setAutoApprove(autoApprove)

- Parameters

Boolean autoApprove - The default value of auto approve setting for this sample definition is set to the desired boolean.

- Returns

Nothing

setComingDueMin(comingDueMin)

Description

Sets the default coming due minutes setting for this sample definition. Allowed locations that belong to this sample definition are initialized with this default setting. The value represents the number of minutes required before a sample is due until the sample is considered coming due. For automatically scheduled samples, they are created prior to actual due time by the number of minutes of this setting.

Syntax

setComingDueMin(comingDueMin)

- Parameters

double comingDueMin - The number of minutes required before a sample is due until it is coming due.

- Returns

None

setDescription(description)

Description

Sets the description of this sample definition.

Syntax

setDescription(description)

- Parameters

String description - The description given to this sample definition.

- Returns

Nothing

setEnabled(enabled)

Description

Sets sample definition enabled state.

Syntax

setEnabled(enabled)

- Parameters

Boolean enabled - Set to True if the sample definition is to be enabled.

- Returns

Nothing

setInterval(interval)

Description

Sets the default interval for automatically scheduled samples. Allowed locations that belong to this sample definition are initialized with this default interval. The units are defined by the Interval type defined for this sample definition.

Syntax

setInterval(interval)

- Parameters

Double interval - The interval to be set for the specified samples.

- Returns

Nothing

setIntervalType(intervalType)

Description

Sets the default interval type for automatically scheduled samples. Allowed locations that belong to this sample definition are initialized with this default interval type. The return value must match those configured on the Quality tab for the enterprise in the Sample Interval list.

Syntax

setIntervalType(intervalType)

- Parameters

String intervalType - The interval type to be set for the specified samples.

- Returns

Nothing

setMeasurementCount(measurementCount)

Description

Sets this number of measurement to be used when creating samples based on the sample definition.

Syntax

setMeasurementCount(count)

- Parameters

Integer measurementCount - The number of measurements associated with this sample definition.

- Returns

Nothing

setName(name)

Description

Sets a name for this sample definition.

It is recommended that once samples have been created using a specific name, it should not be changed.

Syntax

setName(name)

- Parameters

String name - Name to be set to the sample definition.

- Returns

Nothing

setOverdueMin(overdueMin)

Description

Sets the default overdue minutes setting for this sample definition. Allowed locations that belong to this sample definition are initialized with this default setting. The value represents the number of minutes required after a sample is due until the sample is considered overdue.

Syntax

setOverdueMin(overdueMin)

- Parameters

double overdueMin - The value represents the number of minutes required after a sample is due until the sample is considered overdue.

- Returns

Nothing

Attribute properties:

addAttribute(attribute)

Description

Adds a new attribute defined in the attribute parameter.

Syntax

addAttribute(attribute)

- Parameters

SampleDefinitionAttribute attribute - The attribute to be added to the sample definition.

- Returns

String result - Any error messages are returned, otherwise an empty string is returned.

attributeExists(attribute)

Description

Returns true if the attribute specified in the parameter already exists for this sample definition. True will also be returned for disabled attributes.

Syntax

attributeExists(attribute)

- Parameters

SampleDefinitionAttribute attribute - The attribute to check the existence for.

- Returns

boolean True if the specified attribute exists and False otherwise.

clearAttributes()

Description

All attributes contained in this sample definition are removed. Instead of the attributes being permanently removed, thier enabled flag is set to false.

Syntax

clearAttributes()

- Parameters

None

- Returns

Nothing

getAllAttribute()

Description

Returns a list of all attributes associated with this sample definition. This function will return enabled and disabled attributes.

Syntax

getAllAttributes()

- Parameters

None

- Returns

A list of SampleDefinitionAttribute Objects for this sample definition.

getAttribute(id)

Description

Returns the attribute specified by the id parameter.

Syntax

getAttribute(id)

- Parameters

Integer id - The identifier of the sample definition to return for.

- Returns

SampleDefinitionAttribute - The sample definition attribute object specified by the id parameter.

getAttribute(name)

Description

Returns the attribute with the same name as the name parameter.

Syntax

getAttribute(name)

- Parameters

String name - Name of the attribute to be returned for.

- Returns

SampleDefinitionAttribute - The attribute specified by the name parameter.

getEnabledAttributes()

Description

Returns a list of all attributes associated with this sample definition. This function will return only enabled attributes.

Syntax

getEnabledAttribute()

- Parameters

None

- Returns

List < SampleDefinitionAttribute > - A list of all attributes for this sample definition.

removeAttribute(attribute)

Description

Removes the attribute defined in the attribute parameter. Instead of attributes being permanently removed, their enabled flag is set to false. Any error messages are returned, otherwise an empty string is returned.

Syntax

removeAttribute(attribute)

- Parameters

SampleDefinitionAttribute attribute - The attribute to be removed from this sample definition.

- Returns

String result - If the attribute is successfully removed, then a message "Attribute not found" will be returned.

removeAttribute(index)

Description

Removes the attribute defined in the index parameter. Instead of attributes being permanently removed, thier enabled flag is set to false. Any error messages are returned, otherwise an empty string is returned.

Syntax

removeAttribute(index)

- Parameters

int index - The index of the attribute to be removed from this sample definition.

- Returns

String result - If the attribute is successfully removed, then a message "Attribute not found" will be returned.

removeAttribute(name)

Description

Removes the attribute defined in the name parameter. Instead of attributes being permanently removed, thier enabled flag is set to false. Any error messages are returned, otherwise an empty string is returned.

Syntax

removeAttribute(name)

- Parameters

String name - The name of the attribute to be removed from this sample definition.

- Returns

String result - If the attribute is successfully removed, then a message "Attribute not found" will be returned.

Allowed location properties:

addAllowedLocation(location)

Description

Adds a new allowed location defined in the location parameter. By adding an allowed location to this sample definition, this type of sample will appear as an option for the location and the real time location will be saved along with associated samples. For example, shift, product code, ref No and additional factor information is saved along with the sample data.

Syntax

addAllowedLocation(location)

- Parameters

SampleDefinitionLocation location - The location to be added to this sample definition.

- Returns

String result - Any error messages are returned, otherwise an empty string is returned.

allowedLocationExists(location)

Description

Returns true if the allowed location specified in the parameter already exists for this sample definition. True will also be returned for allowed locations that have been removed, but not committed by saving the sample definition.

Syntax

allowedLocationExists(location)

- Parameters

SampleDefinitionLocation location - The location to check the existence for.

- Returns

B oolean True if the location exist and False otherwise.

clearAllowedLocations()

Description

All allowed locations contained in this sample definition are removed. Allowed locations are permanently removed but can be added back. SPC data is not lost and will appear in the control charts and analysis.

Syntax

clearAllowedLocations()

- Parameters

None

- Returns

Nothing

getAllAllowedLocations(includeRemoved)

Description

Returns a list of all allowed locations associated with this sample definition. If the includeRemoved parameter is true, the results will include removed allowed locations that have not been committed by saving the sample definition.

Syntax

getAllAllowedLocations(includeRemoved)

- Parameters

Boolean includeRemoved - Set it to True if the results should include the removed locations and False otherwise.

- Returns

List of Sample Definition Location - A list of all the locations associated with this sample definition is returned.

getAllowedLocation(name)

Description

Returns the allowed location with the same name as the name parameter.

Syntax

getAllowedLocation(name)

- Parameters

String name - The name of the location to return for.

- Returns

SampleDefinitionLocation - The sample definition location object specified by the name parameter.

removeAllowedLocation(index)

Description

Removes the allowed location defined by the index parameter. Allowed locations are permanently removed but can be added back. SPC data is not lost and will appear in the control charts and analysis.

Syntax

removeAllowedLocation(index)

- Parameters

int index - The index of location to be removed for.

- Returns

String result - Any error messages are returned, otherwise an empty string is returned.

removeAllowedLocation(location)

Description

Removes the allowed location defined in the location parameter. Allowed locations are permanently removed but can be added back. SPC data is not lost and will appear in the control charts and analysis.

Syntax

removeAllowedLocation(location)

- Parameters

SampleDefinitionLocation location - The location to be removed from this sample definition.

- Returns

String result - Any error messages are returned, otherwise an empty string is returned.

removeAllowedLocation(locationName)

Description

Removes the allowed location defined by the name parameter. Allowed locations are permanently removed but can be added back. SPC data is not lost and will appear in the control charts and analysis.

Syntax

removeAllowedLocation(locationName)

- Parameters

String locationName - The name of location to be removed for.

- Returns

String result - Any error messages are returned, otherwise an empty string is returned.

Control limit properties:

addControlLimit(controlLimit)

Description

Adds a new control limit defined in the controlLimit parameter. By adding a control limit to this sample definition, it will show as an option in the control charts and may also be used when evaluating signals. The controlLimit parameter must be a valid control limit that appears in the enterprise production item. Any error messages are returned, otherwise an empty string is returned.

Syntax

addControlLimit (controlLimit)

- Parameters

SampleDefinitionControlLimit controlLimit - The control limit to be added.

- Returns

String controlLimit - Sample control limit with the same name or ID already exists.

clearControlLimits()

Description

All control limits contained in this sample definition are removed.

Syntax

clearControlLimits()

- Parameters

None

- Returns

Nothing

controlLimitExists(controlLimit)

Description

Returns true if the control limit specified in the parameter already exists for this sample definition.

Syntax

controlLimitExists(controlLimit)

- Parameters

SampleDefinitionControlLimit contolLimit - The control limit to check for existence.

- Returns

True, if the specified control limit exist.

getAllAllowedLocations(includeRemoved)

Description

Returns a list of all allowed locations associated with this sample definition. If the includeRemoved parameter is true the results will include removed allowed locations that have not been committed by saving the sample definition.

Syntax

getAllAllowedLocations(includeRemoved)

- Parameters

boolean includeRemoved - True if the allowed locations that has been removed to be included.

- Returns

A list of all allowed locations associated with the sample definition.

getAllControlLimits()

Description

Returns all control limits that have been selected for this sample definition.

Syntax

getAllControlLimits()

- Parameters

None

- Returns

A list of all control limits for the sample definition.

getControlLimit(name)

Description

Returns the control limit that has the same name as the name parameter.

Syntax

getControlLimit(name)

- Parameters

String name - Name of the control limit.

- Returns

SampleDefinitionControlLimit control limit - The control limit with the specified name.

removeControlLimit(controlLimit)

Description

Removes the control limit defined in the controlLimit parameter.

Syntax

removeControlLimit (controlLimit)

- Parameters

SampleDefinitionControlLimit controlLimit - The control limit to be removed.

- Returns

Any error messages are returned, otherwise an empty string is returned.

removeControlLimit(controlLimitName)

Description

Removes the control limit defined in the controlLimitName parameter.

Syntax

removeControlLimit (controlLimitName)

- Parameters

String controlLimitName - Name of the control limit to be removed.

- Returns

Any error messages are returned, otherwise an empty string is returned.

removeControlLimit(index)

Description

Removes the control limit defined in the index parameter.

Syntax

removeControlLimit (index)

- Parameters

int index - The index of the control limit to remove for.

- Returns

Any error messages are returned, otherwise an empty string is returned.

Signal properties:

addSignal(signal)

Description

Adds a new signal defined in the signal parameter. By adding a signal to this sample definition, it will show as an option in the control charts and may also be automatically evaluated. The signal parameter must be a valid signal that appears in the enterprise production item.

Syntax

addSignal(signal)

- Parameters

SampleDefinitionSignal signal - The new signal to be added.

- Returns

Any error messages are returned, otherwise an empty string is returned.

clearSignals()

Description

All signals contained in this sample definition are removed.

Syntax

clearSignals()

- Parameters

None

- Returns

Nothing

getAllSignals()

Description

Return all signals that have been selected for this sample definition.

Syntax

getAllSignals()

- Parameters

None

- Returns

A List of all signals for the sample definition.

getSignal(name)

Description

Returns the signal that has the same name as the name parameter.

Syntax

getSignal(name)

- Parameters

String name - The name of the signal to be returned for.

- Returns

SampleDefinitionSignal - The sample definition signal object specified by the name parameter.

removeSignal(index)

Description

Removes the signal defined in the index parameter.

Syntax

removeSignal(index)

- Parameters

int index - The index of the signal to be removed.

- Returns

Any error messages are returned, otherwise an empty string is returned.

removeSignal(signal)

Description

Removes the signal defined in the signal parameter.

Syntax

removeSignal(signal)

- Parameters

SampleDefinitionSignal signal - The signal to be removed.

- Returns

Any error messages are returned, otherwise an empty string is returned.

removeSignal(signalName)

Description

Removes the signal defined in the signalName parameter.

Syntax

removeSignal(signalName)

- Parameters

int signalName - The name of the signal to be removed.

- Returns

Any error messages are returned, otherwise an empty string is returned.

signalExists(signal)

Description

Returns true if the signal specified in the parameter already exists for this sample definition.

Syntax

signalExists(signal)

- Parameters

SampleDefinitionSignal signal - The signal to check the existence for.

- Returns

True, if there exist the specified signal.

Sample Definition Attribute

The sample definition attribute object holds all of the information defining an attribute used in samples. A sample definition attribute specifies the name, data type, default value and more for a single attribute that resides in a sample definition.

Properties:

getDatatype()

Description

Returns the attribute data type for this attribute. See Attribute Data Type for more information.

Syntax

getDatatype()

- Parameters

None

- Returns

AttributeDataType - The attribute data type object for this sample definition attribute.

getDefaultValue()

Description

Returns the default value for this attribute. If this optional default value exists, the sample's measurement values associated with this attribute are automatically set to this value when a sample is created.

Syntax

getDefaultValue()

- Parameters

None

- Returns

Object defaultValue - The value associated with this sample definition attribute. Object type is dependent upon the attribute type for this specific attribute. See Attribute Data Type for more information

getDescription()

Description

Returns the description of this attribute.

Syntax

getDescription()

- Parameters

None

- Returns

String description - The description for this sample definition attribute.

getEnabled()

Description

Returns true if this attribute is enabled. If an attribute is disabled, it will not appear during sample entry. Based on the value of the included disabled attributes property on the SPC Selector component, disabled attributes will not show on the control charts.

Syntax

getEnabled()

- Parameters

None

- Returns

True, if this attribute is enabled.

getFormat()

Description

Returns the format for this attribute. The format is used to verify formatting values on the control charts and that entered data is correctly formatted. See Attribute Data Type for more information.

Syntax

getFormat()

- Parameters

None

- Returns

String format - The format defined for this sample definition attribute.

getId()

Description

Returns the database created ID for this attribute.

Syntax

getId()

- Parameters

None

- Returns

Integer id - The identifier for the specified attribute.

getMaxValue()

Description

Sets the maximum value for this attribute. If this optional maximum value exists, the sample's measurement values associated with this attribute are required to be less than or equal to this value.

Syntax

getMaxValue()

- Parameters

None

- Returns

Object maxValue - The maximum value set for this attribute. Object type is dependent upon the attribute type for this specific attribute. See Attribute Data Type for more information

getMeasurementCount()

Description

Gets the measurement count associated with this sample definition attribute.

Syntax

getMeasurementCount()

- Parameters

None

- Returns

int measurementCount - The measurement count associated with this sample definition attribute.

getMinValue()

Description

Returns the minimum value for this attribute. If this optional minimum value exists, the sample's measurement values associated with this attribute are required to be greater than or equal to this value.

Syntax

getMinValue()

- Parameters

None

- Returns

Object minValue - The minimum value set for this attribute. Object type is dependent upon the attribute type for this specific attribute. See Attribute Data Type for more information

getName()

Description

Returns the name of this attribute.

Syntax

getName()

- Parameters

None

- Returns

String name - Name of this sample definition attribute.

getNew()

Description

Returns a new sample definition attribute instance.

Syntax

getNew()

- Parameters

None

- Returns

SampleDefinitionAttribute - The newly created sample attribute object.

getParent()

Description

Returns the sample definition that this attribute is a child of.

Syntax

getParent()

- Parameters

None

- Returns

SampleDefinition - The sample definition object that is a parent of this attribute.

getRequired()

Description

Returns true if this attribute is required while entering samples. If an attribute is required, a value must be entered before the sample will be saved.

Syntax

getRequired()

- Parameters

None

- Returns

True, if this attribute is required.

getWeight()

Description

Returns the weight for this attribute.

Syntax

getWeight()

- Parameters

None

- Returns

Double weight - The quantity for this attribute.

isModified()

Description

Returns true if this attribute definition has been modified.

Syntax

isModified()

- Parameters

None

- Returns

True, if this attribute definition has been modified.

isNew()

Description

Returns true if this attribute definition is new.

Syntax

isNew()

- Parameters

None

- Returns

True, if this attribute definition is new.

setDatatype(dataType)

Description

Sets this attribute's data type. See Attribute Data Type for more information.

Syntax

setDatatype(dataType)

- Parameters

AttributeDataType dataType - The data type that is to be set for this sample definition attribute.

- Returns

Nothing

setDescription(description)

Description

Sets the description of this attribute.

Syntax

setDescription(description)

- Parameters

String description - The description to be set for this sample definition attribute.

- Returns

Nothing

setEnabled(enabled)

Description

Sets sample definition enabled state.

Syntax

setEnabled(enabled)

- Parameters

boolean enabled - Set this to True if the sample definition is to be enabled and False otherwise.

- Returns

Nothing

setFormat(format)

Description

Sets this attribute's format. The format is used to verify formatting values on the control charts and that entered data is correctly formatted.

Syntax

setFormat(format)

- Parameters

String format - The format to set the sample definition attribute for.

- Returns

Nothing

setDefaultValue(defaultValue)

Description

Sets the default value for this attribute. If this optional default value exists, the sample's measurement values associated with this attribute are automatically set to this value when a sample is created.

Syntax

setDefaultValue(defaultValue)

- Parameters

Object defaultValue - The value stored with this sample definition attribute. Object type is dependent upon the attribute type for this specific attribute. See Attribute Data Type for more information. Use **None** to clear this value.

- Returns

Nothing

setMaxValue(maxValue)

Description

Sets the maximum value for this attribute. If this optional maximum value exists, the sample's measurement values associated with this attribute are required to be less than or equal to this value.

Syntax

setMaxValue(maxValue)

- Parameters

Object maxValue - The maximum value to set the attribute for. Object type is dependent upon the attribute type for this specific attribute. See Attribute Data Type for more information. Use **None** to clear this value.

- Returns

Nothing

setMeasurementCount(measurementCount)

Info

An attribute cannot be set to have a larger measurement count than its definition.

Description

Sets this number of measurement to be used when creating samples based on the sample definition attribute.

Syntax

setMeasurementCount(measurementCount)

- Parameters

int measurementCount - The measurement count to set for.

- Returns

Nothing

setMinValue(minValue)

Description

Sets the minimum value for this attribute. If this optional minimum value exists, the sample's measurement values associated with this attribute are required to be greater than or equal to this value.

Syntax

setMinValue(minValue)

- Parameters

Object minValue - The minimum value to set the attribute for. Object type is dependent upon the attribute type for this specific attribute. See Attribute Data Type for more information. Use **None** to clear this value.

- Returns

Nothing

setName(name)

Description

Sets the name used for this attribute.

It is recommended that once samples have been created using this name, it should not be changed.

Syntax

setName(name)

- Parameters

String name - Name to be set for this sample definition attribute.

- Returns

Nothing

setRequired(enabled)

Description

Sets this attribute required state. If an attribute is required, a value must be entered before the sample will be saved.

Syntax

setRequired(enabled)

- Parameters

boolean enabled - Set this to True if an attribute is required and False otherwise.

- Returns

Nothing

setWeight(weight)

Description

Sets the quantity for this attribute.

Syntax

setWeight(weight)

- Parameters

Double weight - The quantity to set the definition attribute for.

- Returns

Nothing

Format Strings

Format strings are used by the getFormat() and setFormat() methods. They consist of one or more of the characters shown in the table below. For example: the format string ##.0 will round to show one decimal place. Search java DecimalFormat for more information.

Symbol	Description
0	A digit. absent digits show as zero
#	A digit, zero shows as absent
.	Placeholder for decimal separator
,	Placeholder for grouping separator
E	Separates mantissa and exponent for exponential formats

;	Separates formats
-	Default negative prefix
%	Multiply by 100 and show as percentage
?	Multiply by 1000 and show as per mille
¤	Currency sign; replaced by currency symbol; if doubled, replaced by international currency symbol; if present in a pattern, the monetary decimal separator is used instead of the decimal separator
X	Any other characters can be used in the prefix or suffix
'	Used to quote special characters in a prefix or suffix

Value	Pattern	Output
123456.789	###,###.###	123,456.789
123456.789	###.##	123456.79
123.78	000000.000	000123.780
12345.67	$###,###.###	$12,345.67
12345.67	\u00A5###,###.###	¥12,345.67

Sample Definition Control Limit

The sample definition control limit object holds all of the information defining a control limit that is applied to a sample definition. Be sure not to confuse a control limit defined in the Ignition designer with the sample definition control limit object. The sample definition control limit object connects a control limit defined in the Ignition designer with a sample definition.

Once a control limit is associated with a sample definition, it will appear as an option in the SPC Selector and can appear on control charts. It will also be included during automatic signal evaluations that require the control limit.

Properties:

getEnabled()

Description

Returns true if this sample definition control limit is enabled. If disabled, it will not show as an option on the control charts.

Syntax

getEnabled()

- Parameters

None

- Returns

True if this sample definition control limit is enabled.

getGroup()

Description

Syntax

getGroup()

- Parameters

None

- Returns

String

getId()

Description

Returns the database created ID for this sample definition control limit.

Syntax

getId()

- Parameters

None

- Returns

Integer id - The identifier for this sample definition control limit.

getKind()

Description

Returns the kind of control limit. There are different types of control limits and calculations for each type of chart category and this property makes this association between the two.

Syntax

getKind()

- Parameters

None

- Returns

ControlLimitKindTypes kind - The type of this control limit.

getKindAsInt()

Description

Syntax

getKindAsInt()

- Parameters

None

- Returns

Integer

getName()

Description

Returns the name of this control limit as defined in the Ignition designer.

Syntax

getName()

- Parameters

None

- Returns

String name - The name of this control limit.

getParent()

Description

Returns the sample definition that this control limit is a child of.

Syntax

getParent()

- Parameters

None

- Returns

SampleDefinition

isModified()

Description

Returns true if this sample definition control limit has been modified.

Syntax

isModified()

- Parameters

None

- Returns

True if this sample definition control limit has been modified.

isNew()

Description

Returns true if this sample definition control limit is new.

Syntax

isNew()

- Parameters

None

- Returns

True, if this sample definition control limit is new.

resetModified()

Description

This script function will undo the modifications done.

Syntax

resetModified()

- Parameters

None

- Returns

Nothing

setEnabled(enabled)

Description

Sets this sample definition control limit enabled state. If disabled, it will not show as an option on the control charts.

Syntax

setEnabled(enabled)

- Parameters

boolean enabled

- Returns

Nothing

setGroup(group)

Description

Syntax

setGroup(group)

- Parameters

String group

- Returns

Nothing

setKind(kind)

Description

Sets the kind of control limit. There are different types of control limits and calculations for each type of chart category and this property makes this association between the two.

Syntax

setKind(kind)

- Parameters

ControlLimitKindTypes kind - The type of this control limit.

- Returns

Nothing

setKind(ordinal)

Description

Set the kind of control limit based on a ControlLimitKindTypes ordinal value. There are different types of control limits and calculations for each type of chart category and this property makes this association between the two.

Syntax

setKind(ordinal)

- Parameters

int ordinal

- Returns

Nothing

setName(name)

Description

Sets the name of this control limit as defined in the Ignition designer.

Syntax

setName(name)

- Parameters

String name - The name of this control limit.

- Returns

Nothing

setParent(parent)

Description

Syntax

setParent(parent)

- Parameters

SampleDefinition parent

- Returns

Nothing

Sample Definition Location

The sample definition location object holds all of the information defining a location that samples are taken from. A sample definition location may specify the interval to schedule samples and various due time values. Be sure not to confuse a production location with the sample definition location object. The sample definition location object defines a production location that samples for a sample definition can be taken from.

When using the term Location within the SPC module, it refers to a virtual location where actual samples are taken. For example, if a sample bottle is taken from packaging line 1 and is tested in the lab for color, then the location is packaging line 1. In addition to the lab taking samples from this location, the operator can take samples to test labels. The tag property is used to define the ownership of who is responsible to take a sample.

Properties:

getAutoApprove()

Description

Returns the auto approve setting for this location.

Syntax

getAutoApprove()

- Parameters

None

- Returns

If true, samples will be automatically approved when they are recorded. If false, they have to be manually approved.

getComingDueMin()

Description

Returns the coming due minutes setting for this location. The value represents the number of minutes required before a sample is due until the sample is considered coming due. For automatically scheduled samples, they are created prior to actual due time by the number of minutes of this setting.

Syntax

getComingDueMin()

- Parameters

None

- Returns

Double comingDueMin - The coming due in minutes for this sample definition location.

getDuration()

Description

Returns the number of minutes needed to take a sample for this location.

Syntax

getDuration()

- Parameters

None

- Returns

double duration - Time duration in minutes to take the sample at this specified location.

getEnabled()

Description

Returns true if this sample definition location is enabled. If disabled, samples will not be automatically scheduled or appear in sample definition selection lists for the production location.

Syntax

getEnabled()

- Parameters

None

- Returns

True, if this sample definition location is enabled.

getId()

Description

Returns the database created ID for this sample definition location.

Info

This is not the same as the production location ID.

Syntax

getId()

- Parameters

None

- Returns

int id - The ID for this sample definition location.

getInterval()

Description

Returns the interval for automatically scheduling samples for this location. The units are defined by the Interval type defined for this sample definition.

Syntax

getInterval()

- Parameters

None

- Returns

double interval

getIntervalType()

Description

Returns the interval type for automatically scheduling samples for this location. The return value must match those configured on the Quality tab for the enterprise in the Sample Interval list.

Syntax

getIntervalType()

- Parameters

None

- Returns

String intervalType

getLocationID()

Description

Returns the database created ID for the production location that this sample definition location is associated with.

Info

This is the same as the production location ID.

Syntax

getLocationID()

- Parameters

None

- Returns

int locationID - An integer value representing the ID for the location.

getName()

Description

Returns the name of the production location associated with this sample definition location. This name also appears as the name for this sample definition location.

Syntax

getName()

- Parameters

None

- Returns

String name - Name of the sample definition location.

getNew(locationID, name)

Description

Returns a new sample definition location instance for the production location specified by the locationID parameter. The new instance name is specified by the name parameter.

Syntax

getNew(locationID, name)

- Parameters

int locationID - An integer value representing the ID for the location.

String name - Name of the new sample definition location.

- Returns

The newly created SampleDefinitionLocation object for the specified production location.

getOverdueMin()

Description

Returns the overdue minutes setting for this location. The value represents the number of minutes required before a sample is due until the sample is considered overdue.

Syntax

getOverdueMin()

- Parameters

None

- Returns

Overdue in minutes for this sample definition location.

getParent()

Description

Returns the sample definition that this location is a child of.

Syntax

getParent()

- Parameters

None

- Returns

SampleDefinition parent - Sample definition object which is the parent to this location.

getTag()

Description

Returns the tag setting for this location.

Syntax

getTag()

- Parameters

None

- Returns

The tag for this sample definition location.

isModified()

Description

Returns true if this sample definition has been modified.

Syntax

isModified()

- Parameters

None

- Returns

True, if this sample definition has been modified.

isNew()

Description

Returns true if this sample definition is new.

Syntax

isNew()

- Parameters

None

- Returns

True, if this sample definition is new.

setAutoApprove(autoApprove)

Description

Sets the auto approve setting for this location. If true, samples will be automatically approved when they are recorded. If false, they have to be manually approved.

Syntax

setAutoApprove(autoApprove)

- Parameters

boolean autoApprove - True if the sample should be automatically approved, False otherwise.

- Returns

Nothing

setComingDueMin(comingDueMin)

Description

Sets the coming due minutes setting for this location. The value represents the number of minutes required before a sample is due until the sample is considered coming due. For automatically scheduled samples, they are created prior to actual due time by the number of minutes of this setting.

Syntax

setComingDueMin(comingDueMin)

- Parameters

double comingDueMin - The coming due in minutes for this sample definition location.

- Returns

Nothing

setDuration(duration)

Description

Sets the number of minutes needed to take a sample for this location.

Syntax

setDuration(duration)

- Parameters

double duration - Time duration in minutes to take the sample at this specified location.

- Returns

Nothing

setEnabled(enabled)

Description

Sets sample definition location enabled state. If disabled, samples will not be automatically scheduled or appear in sample definition selection lists for the production location.

Syntax

setEnabled(enabled)

- Parameters

boolean enabled - True if the sample should be automatically scheduled, False otherwise.

- Returns

Nothing

setInterval(interval)

Description

Sets the interval for automatically scheduling samples for this location. The units are defined by the Interval type defined for this sample definition.

Syntax

setInterval(interval)

- Parameters

double interval

- Returns

Nothing

setIntervalType(intervalType)

Description

Sets the interval type for automatically scheduling samples for this location. The intervalType value must match those configured on the Quality tab for the enterprise in the Sample Interval list.

Syntax

setIntervalType(intervalType)

- Parameters

String intervalType

- Returns

Nothing

setLocationID(locationID)

Description

Info

This is the same as the production location ID.

Syntax

setLocationID(locationID)

- Parameters

int locationID - An integer value representing the ID for the location.

- Returns

Nothing

setOverdueMin(overdueMinutes)

Description

Sets the overdue minutes setting for this location. The value represents the number of minutes required before a sample is due until the sample is considered overdue.

Syntax

setOverdueMin(overdueMinutes)

- Parameters

double overdueMinutes

- Returns

Nothing

setParent(parent)

Description

Syntax

setParent(parent)

- Parameters

SampleDefinition parent - Sample definition object which is the parent to this location.

- Returns

Nothing

setTag(tag)

Description

Sets the tag setting for this location. The tag is used to assign ownership of who is responsible to take samples. For example, set to “Lab" if the lab is responsible or “Operator” if the operator is responsible.

Syntax

setTag(tag)

- Parameters

String tag - Name of the tag to be set.

- Returns

Nothing

Sample Definition Signal

The sample definition signal object holds all of the information defining a signal that will be applied to a sample definition. Be sure not to confuse a signal defined in the Ignition designer with the sample definition signal object. The sample definition signal object connects a signal defined in the Ignition designer with a sample definition.

If a signal is associated with a sample definition, it will appear as an option in the SPC Selector and can appear on control charts. It will also be included during automatic signal evaluations.

Properties:

getEnabled()

Description

Returns true if this sample definition signal is enabled. If disabled, it will not show as an option on the control charts.

Syntax

getEnabled()

- Parameters

None

- Returns

True, if this sample definition signal is enabled.

getId()

Description

Returns the database created ID for this sample definition signal.

Syntax

getId()

- Parameters

None

- Returns

int id - Integer value representing the id of this sample definition signal.

getKind()

Description

Returns the the kind of signal. There are different types of signals and calculations for each type of chart category and this property makes this association between the two.

Syntax

getKind()

- Parameters

None

- Returns

SignalKindTypes

getName()

Description

Returns the name of this signal as defined in the Ignition designer.

Syntax

getName()

- Parameters

None

- Returns

String name

getParent()

Description

Returns the sample definition that this signal is a child of.

Syntax

getParent()

- Parameters

None

- Returns

SampleDefinition

isModified()

Description

Returns true if this sample definition signal has been modified.

Syntax

isModified()

- Parameters

None

- Returns

True, if this sample definition signal is modified.

isNew()

Description

Returns true if this sample definition signal is new.

Syntax

isNew()

- Parameters

None

- Returns

True, if this sample definition signal is new.

setEnabled(enabled)

Description

Sets this sample definition signal enabled state. If disabled, it will not show as an option on the control charts.

Syntax

setEnabled(enabled)

- Parameters

boolean enabled

- Returns

Nothing

setKind(kind)

Description

Sets the kind of signal. There are different types of signals and calculations for each type of chart category and this property makes this association between the two.

Syntax

setKind(kind)

- Parameters

SignalKindTypes kind

- Returns

Nothing

setKind(ordinal)

Description

Sets the kind of signal based on a SignalKindTypesordinal value. There are different types of signals and calculations for each type of chart category and this property makes this association between the two.

Syntax

setKind(ordinal)

- Parameters

int ordinal

- Returns

Nothing

setName(name)

Description

Sets the name of this signal as defined in the Ignition designer.

Syntax

setName(name)

- Parameters

String name

- Returns

Nothing

Sample States

This object contains the state of the sample. Various states are unknown, overdue, due, coming due, waiting approval, approved and excluded. The default integer corresponding to these states are -1, 0, 1, 2, 3, 4, 5 respectively.

addState(state)

Description

Adds a state.

Syntax

addState(state)

- Parameters

Integer state - The integer representing state of the sample to add.

- Returns

Nothing

- Scope

All

getState()

Description

Returns the integer corresponding to state of this sample.

Syntax

getState()

- Parameters

None

- Returns

Integer state - The integer representing the state of sample.

- Scope

All

hasState()

Description

Checks the existence of the given state.

Syntax

hasState()

- Parameters

Integer state - The state of the sample.

- Returns

boolean state - True if there exist the sample represented by the given parameter.

- Scope

All

removeState(state)

Description

Removes a state.

Syntax

removeState(state)

- Parameters

Integer state - The integer representing state of the sample to remove.

- Returns

Nothing

- Scope

All

Properties:

getTypeFromDescription(description)

Description

Syntax

getTypeFromDescription(description)

- Parameters

String description - Description of the signal auto evaluated period type to return for.

- Returns

SignalAutoEvaluatePeriodTypes

getTypeFromName(name)

Description

Syntax

getTypeFromName(name)

- Parameters

String name - Name of the signal auto evaluated period type to return for.

- Returns

SignalAutoEvaluatePeriodTypes

intToType(ordinal)

Description

Returns the SignalAutoEvaluatePeriodTypes object for the ordinal value specified.

Syntax

intToType(ordinal)

- Parameters

int ordinal - Integer indicating position of the period type.

- Returns

SignalAutoEvaluatePeriodTypes

Signal Event

The event is created to get a signal information.

getCalcValues()

Description

Returns the value information of the signal.

Syntax

getCalcValues()

- Parameters

None

- Returns

SPCCalcValueCollection calcValues - The value of the signal.

- Scope

All

getData()

Description

Gets the SPC data.

Syntax

getData()

- Parameters

None

- Returns

AnalysisDataset data - The SPC data associated with the signal.

- Scope

All

getSampleSize()

Description

Returns the sample size.

Syntax

getSampleSize()

- Parameters

None

- Returns

Integer sampleSize - The size of the sample.

- Scope

All

getSignalName()

Description

Returns the signal name.

Syntax

getSignalName()

- Parameters

None

- Returns

String signalName - The name of the signal.

- Scope

All

Signal Kind Types

The signal kind type object contains the available types that a signal can be:

Available data types:

XBAR

RANGE

SBAR

INDIVIDUAL

MEDIAN

P

NP

U

C

HISTOGRAM

PARETO

MR

Properties:

getCategory()

Description

Returns the category of chart. See SPC Category Types for more information.

Syntax

getCategory()

- Parameters

None

- Returns

SPCCategoryTypes - The category object for the signal kind is returned.

- Scope

All

getDataFormat()

Description

Returns the category of chart. See SPC Category Types for more information.

Syntax

getDataFormat()

- Parameters

None

- Returns

SPCDataFormat

- Scope

All

getText()

Description

Returns the user friendly localized text for the signal kind.

Syntax

getText()

- Parameters

None

- Returns

String text - The localized text for the corresponding signal kind.

- Scope

All

getTypeFromDescription(description)

Description

Syntax

getTypeFromDescription(description)

- Parameters

String description - Description of the signal kind type to get the type for.

- Returns

SignalKindTypes type - Type of signal kind specified by the description parameter.

- Scope

All

getTypeFromName(name)

Description

Returns the signal kind type object for the name value specified.

Syntax

getTypeFromName(name)

- Parameters

String name - Name of the signal kind type.

- Returns

SignalKindTypes type - Type of signal kind specified by the name parameter.

- Scope

All

intToType(ordinal)

Description

Returns the signal kind type object for the ordinal value specified.

Syntax

intToType(ordinal)

- Parameters

int ordinal - Integer indicating position of the signal kind object.

- Returns

SignalKindTypes type - The signal kind type object for the ordinal specified.

- Scope

All

isUserSelectable()

Description

Returns true if the signal kind can be selected by the user.

Syntax

isUserSelectable()

- Parameters

None

- Returns

boolean - True if the signal kind types can be selected by the user, False otherwise.

- Scope

All

SPC Category Types

The SPC category type defines the possible types of charts currently supported by the SPC module.

Available data types:

XBAR

RANGE

SBAR

INDIVIDUAL

MEDIAN

P

NP

U

C

HISTOGRAM

PARETO

MR

Properties:

getShape()

Description

Returns the SPC chart shape type object for the name specified.

Syntax

getShape()

- Parameters

None

- Returns

Shape shape - The shape of the SPC chart.

getTypeFromDescription(description)

Description

Returns the SPC chart shape type object for the description specified.

Syntax

getTypeFromDescription(description)

- Parameters

String description - The note to return the type for.

- Returns

SPCChartShapeTypes type - The SPC chart shape type object specified by the parameter.

getTypeFromName(name)

Description

Returns the SPC chart shape type object for the name specified.

Syntax

getTypeFromName(name)

- Parameters

String name - The name to return the type for.

- Returns

SPCChartShapeTypes type - The SPC chart shape type object specified by the parameter.

intToType(ordinal)

Description

Returns the SPC chart shape type object for the ordinal value specified.

Syntax

intToType(ordinal)

- Parameters

int ordinal - Integer indicating position of the shape type.

- Returns

SPCChartShapeTypes type - The SPC chart shape type object specified by the parameter.

isFilled()

Description

Boolean indicating if this chart shape is of filled type or not. The various filled types are Diamond-filled, Dot-filled, Rectangle-filled.

Syntax

isFilled()

- Parameters

None

- Returns

boolean filled - True if this chart shape is of filled type and False otherwise.

Properties:

getDefName()

Description

Gets the name of this SPC definition.

Syntax

getDefName()

- Parameters

None

- Returns

String defName - The name of this SPC definition.

getDefUUID()

Description

Gets the uuid of this SPC definition.

Syntax

getDefUUID()

- Parameters

None

- Returns

String defUUID - The unique identifier corresponding to this SPC definition.

getInspectedAttrCount()

Inspected Attribute Count

Attribute can contain a counting number (1, 2, 3, 4, …) and represents a number of items inspected for a attribute samples. This attribute data type is recognized and required by the p, np, c and u control charts.

Description

Gets the inspected attribute count associated with this SPC definition.

Syntax

getInspectedAttrCount()

- Parameters

None

- Returns

Integer inspectedAttrCount - The number of inspected items with this SPC definition.

getMeasCount()

Description

Returns the measurement count for this definition.

Syntax

getMeasCount()

- Parameters

None

- Returns

Integer measCount - The measurement count for this SPC definition.

getNonconformingAttrCount()

Non Conforming Attribute Count

Attribute can contain a counting number (1, 2, 3, 4, …) and represents a number of nonconforming items (defective items) for a attribute samples. This attribute data type is recognized and required by the p and np control charts.

Description

Gets the non conforming attribute count for this SPC definition.

Syntax

getNonconformingAttrCount()

- Parameters

None

- Returns

Integer nonconformingAttrCount - The number of non conforming items associated with this SPC definition.

getNonconformityAttrCount()

Non Conformity Attribute Count

Attribute can contain a counting number (1, 2, 3, 4, …) and represents the number of nonconformities items that have (deformities) for a attribute samples. This attribute data type is recognized and required by the c and u control charts.

Description

Gets the non conformity attribute count for this SPC definition.

Syntax

getNonconformityAttrCount()

- Parameters

None

- Returns

Integer nonconformityAttrCount - The number of non conformities items associated with this SPC definition.

isValid()

Description

Boolean indicating if this SPC definition is valid or not.

Syntax

isValid()

- Parameters

None

- Returns

boolean True, if this definition is valid and False otherwise.

setDefUUID(defUUID)

Description

Sets the unique identifier for this SPC definition.

Syntax

setDefUUID(defUUID)

- Parameters

String defUUID - The uuid for this SPC definition.

- Returns

Nothing

setInspectedAttrCount(inspectedAttrCount)

Inspected Attribute Count

Attribute can contain a counting number (1, 2, 3, 4, …) and represents a number of items inspected for a attribute samples. This attribute data type is recognized and required by the p, np, c and u control charts.

Description

Sets the inspected attribute count for this SPC definition.

Syntax

setInspectedAttrCount(inspectedAttrCount)

- Parameters

Integer inspectedAttrCount - The number of inspected items to set for.

- Returns

Nothing

setMeasCount(measCount)

Description

Sets the measurement count for this SPC definition.

Syntax

setMeasCount(measCount)

- Parameters

Integer measCount - The measurement count to set for.

- Returns

Nothing

setNonconformingAttrCount(nonconformingAttrCount)

Non Conforming Attribute Count

Attribute can contain a counting number (1, 2, 3, 4, …) and represents a number of nonconforming items (defective items) for a attribute samples. This attribute data type is recognized and required by the p and np control charts.

Description

Sets the non conforming attribute count for this SPC definition.

Syntax

setNonconformingAttrCount(nonconformingAttrCount)

- Parameters

Integer nonconformingAttrCount - The number of non conforming items to set for.

- Returns

Nothing

setNonconformityAttrCount(nonconformityAttrCount)

Non Conformity Attribute Count

Attribute can contain a counting number (1, 2, 3, 4, …) and represents a number of nonconformities items (defective items) for a attribute samples. This attribute data type is recognized and required by the p and np control charts.

Description

Sets the non conformity attribute count for this SPC definition.

Syntax

setNonconformityAttrCount(nonconformityAttrCount)

- Parameters

Integer nonconformityAttrCount - The number of non conformities items to set for.

- Returns

Nothing

Properties:

createSettings(definitionName, attribute, filters, controlLimits, signals, dataFormatName)

Description

Create a new instance of a SPCSettings object based on the parameters.

Syntax

createSettings(definitionName, attribute, filters, controlLimits, signals, dataFormatName)

- Parameters

String definitionName - The sample definition name for the new settings.

String attribute - The attribute name for the new settings.

String filters - The filters for the new settings. Multiple filter expressions can be separated by commas.

String controlLimits - The control limits for the new settings. Multiple control limits can be separated by commas.

String signals - The SPC rules (signals) for the new settings. Multiple SPC rules can be separated by commas.

String dataFormatName - The data format (control chart type) for the new settings.

- Returns

A new instance of a SPCSettings object.

decodeFilters(filterList)

Description

Decode a list of SPC filter expressions into a java Map object. Each filter key can have multiple filter values.

Syntax

decodeFilters(filterList)

- Parameters

An instance of a java List object containing SPC filter expression strings. Example: "Location=Enterprise\Site\Area\Quality Test Station 1, Location=Enterprise\Site\Area\Quality Test Station 2, Product Code=DEF"

- Returns

An instance of a java Map. The map key is the filter name. For example, "Location" or "Product Code". The value for the key contains a java List object containing all of the filter values. For example, the key "Location" can have the filters values of "Quality Station 1" and "Quality Station 2".

decodeList(input)

Description

Decode a string that can represent a SPC filter, control limits, SPC rules (signals), etc. into a java List object. The input string will be parsed on wither the comma or pipe (|) character and each parsed result will be added to the returned List object.

Syntax

decodeList(input)

- Parameters

The string value to parse.

- Returns

A java List object containing the parsed strings.

decodeParams(optionalParams)

Description

Decode a list of SPC parameters into a java Map object. Each parameter key can have only one parameter value.

Syntax

decodeParams(optionalParams)

- Parameters

A string containing optional parameters separated by either the comma or pipe (|) characters. Example: "PaddingBarCount=4,RowLimit=100,DataBarCount=7, IncludeDisabledAttributes=true"

- Returns

An instance of a java Map containing key value pairs.

encodeList(list)

Description

Encode the specified java List into a single string separated by the pipe (\) character.

Syntax

encodeList(list)

- Parameters

An instance of a java List object containing string values.

- Returns

A single string containing all of the items from the list.

formatDate(date)

Description

Returns a string for the specified date to is formatted correctly for a filter expression.

Syntax

formatDate()

- Parameters

A Date object to format.

- Returns

A SPC settings formatted date string.

getAdditionalFactors()

Description

Gets the additional factors defined with this SPC settings.

Syntax

getAdditionalFactors()

- Parameters

None

- Returns

String additionalFactors - The additional factors defined with this SPC settings.

getAttribute()

Description

Returns the attribute for this settings.

Syntax

getAttribute()

- Parameters

None

- Returns

String attribute - The attribute associated with this SPC settings.

getControlLimits()

Description

Returns the control limits for this SPC settings.

Syntax

getControlLimits()

- Parameters

None

- Returns

String controlLimits - The control limits associated with this setting.

getDataFormat()

Description

Returns the data format associated with this settings.

Syntax

getDataFormat()

- Parameters

None

- Returns

SPCDataFormat - The data format associated with this setting.

getDefinition()

Description

Gets the SPC definition.

Syntax

getDefinition()

- Parameters

None

- Returns

SPCDefinition - The definition corresponding to this settings.

getErrorMessage()

Description

Returns an error message if any.

Syntax

getErrorMessage()

- Parameters

None

- Returns

String message - Any error messages are returned, otherwise an empty string is returned.

getFilters()

Description

Gets the filters of the specified SPC settings.

Syntax

getFilters()

- Parameters

None

- Returns

String filters - The filters associated with this SPC settings.

getId()

Description

Gets the identifier for this SPC setting.

Syntax

getId()

- Parameters

None

- Returns

Integer id - The identifier for this setting.

getMeasurement()

Description

Gets the measurement.

Syntax

getMeasurement()

- Parameters

None

- Returns

String measurement - The measurement corresponding to this setting.

getNonconformingFilter()

Description

Gets the non conforming filter for the setting.

Syntax

getNonconformingFilter()

- Parameters

None

- Returns

String nonconformingFilter - The non conforming filter for this setting.

getOptionalParams()

Description

Returns the optional parameters for this setting.

Syntax

getOptionalParams()

- Parameters

None

- Returns

String optionalParams - The optional parameters for this setting.

getSignals()

Description

Gets the signals associated with this settings.

Syntax

getSignals()

- Parameters

None

- Returns

String signals - The signals associated with this settings.

hasError()

Description

Checks if there is any error in the settings.

Syntax

hasError()

- Parameters

None

- Returns

boolean True if there is any error and False otherwise.

setAdditionalFactors(additionalFactors)

Description

Sets the additional factors for the SPC settings.

Syntax

setAdditionalFactors(additionalFactors)

- Parameters

String additionalFactors - The additional factors to set for.

- Returns

Nothing

setAttribute(attribute)

Description

Sets the attribute for this setting.

Syntax

setAttribute(attribute)

- Parameters

String attribute - The attribute to set for.

- Returns

Nothing

setControlLimits(controlLimits)

Description

Sets the control limits for this SPC setting.

Syntax

setControlLimits(controlLimits)

- Parameters

String controlLimits - The control limits to set for.

- Returns

Nothing

setDataFormat(dataFormat)

Description

Sets the data format for this SPC setting.

Syntax

setDataFormat(dataFormat)

- Parameters

SPCDataFormat - The data format to set for.

- Returns

Nothing

setDefinition(definition)

Description

Sets the SPC definition.

Syntax

setDefinition(definition)

- Parameters

SPCDefinition - The definition to set for.

- Returns

Nothing

setErrorMessage(errorMessage)

Description

Sets the error message for this SPC setting.

Syntax

setErrorMessage(errorMessage)

- Parameters

String errorMessage - The error message to set for.

- Returns

Nothing

setFilters(filters)

Description

Sets the filters for this SPC setting.

Syntax

setFilters(filters)

- Parameters

String filters - The filters to set for.

- Returns

Nothing

setId(id)

Description

Sets the id for this setting.

Syntax

setId(id)

- Parameters

Integer id - The id to set for.

- Returns

Nothing

setMeasurement(measurement)

Description

Sets the measurement for the SPC setting.

Syntax

setMeasurement(measurement)

- Parameters

String measurement - The measurement to set for.

- Returns

Nothing

setNonconformingFilter(filter)

Description

Sets the non conforming filter for the settings.

Syntax

setNonconformingFilter(filter)

- Parameters

String nonconformingFilter - The filter to set for.

- Returns

Nothing

setOptionalParams(optionalParams)

Description

Sets the optional parameters for this SPC settings.

Syntax

setOptionalParams(optionalParams)

- Parameters

String optionalParams - The optional parameters to set for.

- Returns

Nothing

setParetoFilter(paretoFilter)

Description

Sets the pareto filter for this SPC settings.

Syntax

setParetoFilter(paretoFilter)

- Parameters

String paretoFilter - The filter to set for.

- Returns

Nothing

setSignals(signals)

Description

Set signals for this SPC setting.

Syntax

setSignals(signals)

- Parameters

String signals - Signals to set for.

- Returns

Nothing

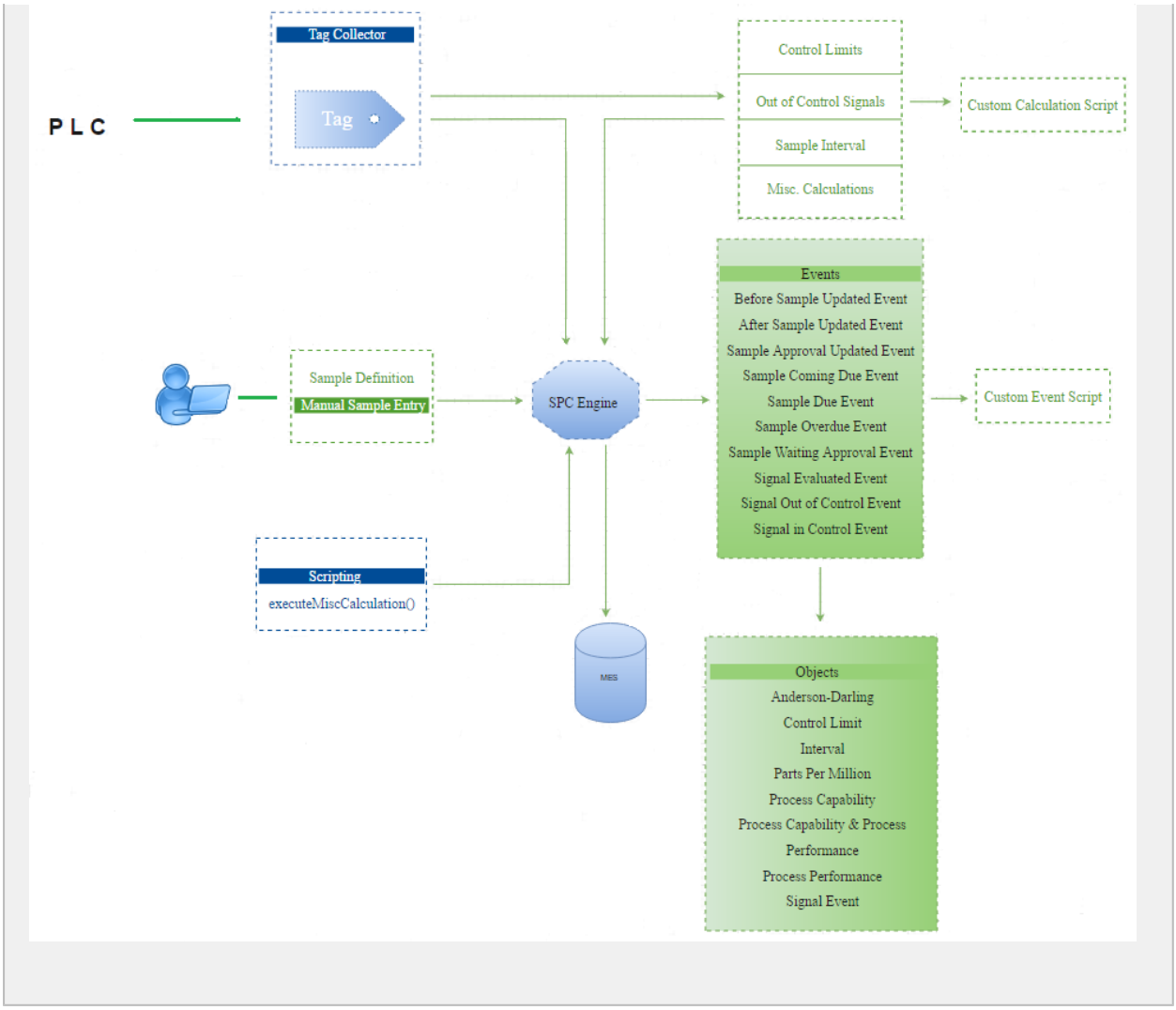
SPC Object Events

The SPC module can capture sample data in a number of different ways...

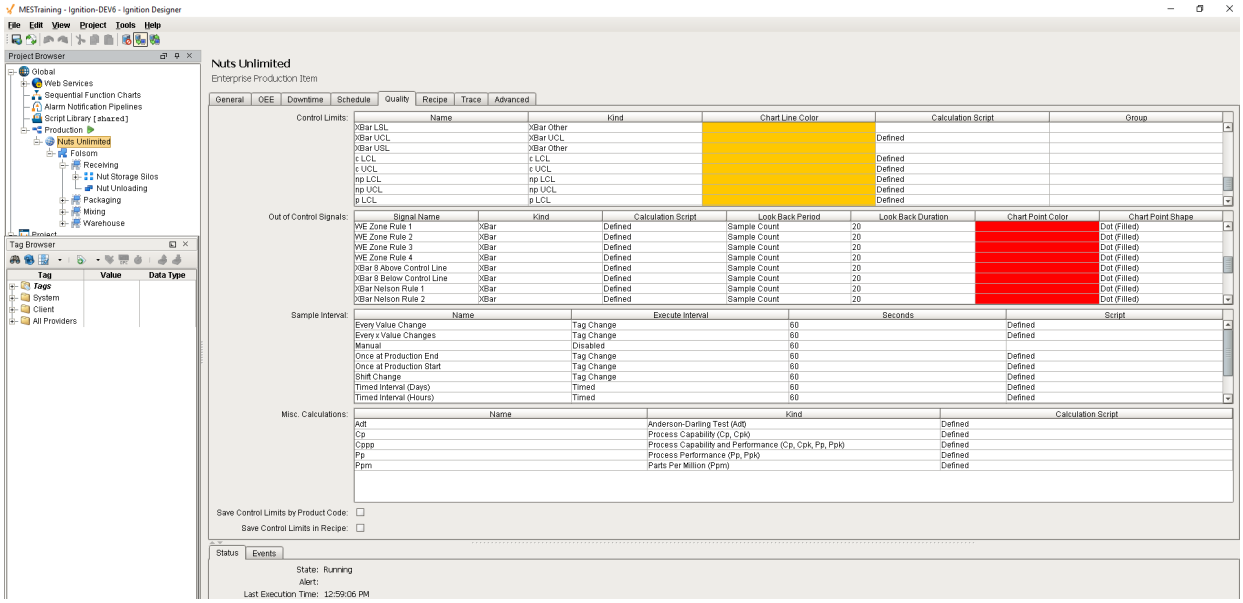
- Tags can be configured to automatically capture sample data by using the Automatic Tag Sample Collector.
- Sample definitions can be created in a Sample Definitions screen that uses the quality components that allows the manual entry of sample values.
- Scripting can be employed to create samples.

However samples are introduced, the SPC engine will perform the necessary calculations based on the Control Limits, Out of Control Signals and Misc. Calculations that have been defined and enabled for that sample.

The SPC Engine will generate an event when a sample point is received and a calculation is performed. Custom scripting can be added to the default calculation event script.



The SPC events can be viewed in the quality tab of the production model in designer at the enterprise level.Whenever an event is fired, a Calculation Kind Types object is passed to the event. You can use this object to get information about the event and the object. SPC object events also allows user to add custom scripts and events in the same way as the MES object events. For more information custom scripting read the Object events section.



Miscellaneous Calculations Event

Event	Description
Anderson-Darling	The event is created to calculate an Anderson-Darling test.
Control Limit	The event is created to calculate a control limit value.
Interval	The event is created to execute the auto scheduling of samples.
Parts Per Million	The event is created to calculate Parts Per Million (PPM).
Process Capability	The event is created to calculate a process capability.
Process Capability & Process Performance	The event is created to calculate a process capability and a process performance.
Process Performance	The event is created to calculate a process performance.
Signal Event	The event is created to get a signal information.

References

system.quality.sample.data.executeMiscCalculation

Advanced SPC Events

The advanced SPC events can be viewed in the advanced tab of the production model in designer at the location level. The following events can be used to change the default handling of samples and signals.

Before Sample Update Event

After Sample Update Event

Sample Approval Updated Event

Sample Coming Due Event

Sample Due Event

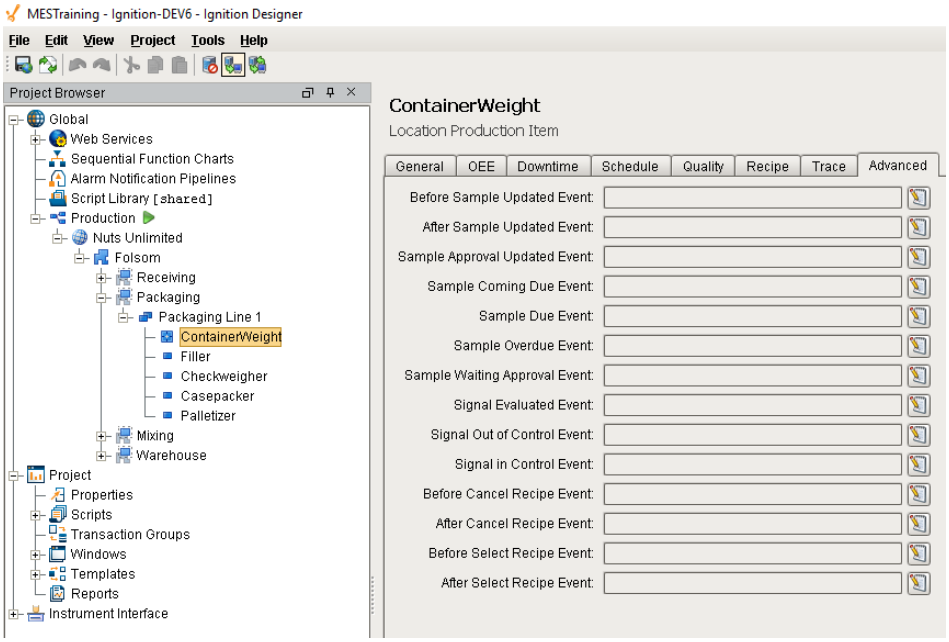
Sample Overdue Event

Sample Waiting Approval Event

Signals Evaluated Event

Signal Out of Control Event

Signal In Control Event



Production Location Events

The following events are by location, which allows for the changing of default handling samples and detection of out of control signals by individual location. Individual handling based on the sample or other criteria, must be done in the script.

In situations where the default handling does not fit the production environment requirements, these events are flexible enough to allow a method to implement exactly what is needed.

After Sample Update Event

After a new sample is added or an existing sample is updated to the database, any script in this event is run. This includes samples that have been scheduled with no measurement data. It is provided to allow for the performance of other actions when sample information changes.

Event properties:

getSample()

Description

Returns the new or updated sample. (See Sample section for more information).

Syntax

getSample()

- Parameters

None

- Returns

Sample sample - The new or updated sample.

Code Example

```
#Add 1 to an unrelated SQLTag value
val = system.tag.getTagValue('[Default]Quality\Test\After Sample
Updated')
val = val + 1
system.tag.writeToTag('[Default]Quality\Test\After Sample Updated'
, val, 1)
```

Before Sample Update Event

Before a new sample is added or an existing sample is updated to the database, any script in this event is run. This includes samples that have been scheduled with no measurement data. It is provided to allow for the addition of more information, performing other actions or preventing the saving of the sample.

Event properties:

getSample()

Description

Returns the new or updated sample. (See Sample section for more information).

Syntax

getSample()

- Parameters

None

- Returns

Sample sample - The new or updated sample.

setCancelUpdate(cancelUpdate)

Description

Used to prevent the sample from being added or updated.

Syntax

setCancelUpdate(cancelUpdate)

- Parameters

boolean cancelUpdate - The default is false, meaning the sample will be added or updated. It is provided to override the default adding or updating of samples and should be used with caution.

- Returns

Nothing

isCancelUpdate(cancelUpdate)

Description

Returns the current state of the cancel update flag.

Syntax

isCancelUpdate()

- Parameters

None

- Returns

boolean cancelUpdate - True, if the current state of the cancel update flag is true and False otherwise.

Code Example

```
#Add 1 to an unrelated SQLTag value
val = system.tag.getTagValue('[Default]Quality\Test\Before Sample
Updated')
val = val + 1
system.tag.writeToTag('[Default]Quality\Test\Before Sample
Updated', val, 1)


#Get the sample from the event
sample = event.getSample()


#Access the additional factors from the sample
addlFactors = sample.getAllAddlFactors()
if len(addlFactors) > 0:
 print "%d additional factors exist." % (len(addlFactors))


 print "val = %d, sampleUUID = %s" % (val, sample.getSampleUUID())
```

Control Limit Result

getLimitName()

Description

Returns the name of the control limit.

Syntax

getLimitName()

- Parameters

None

- Returns

String limitName - Name of the control limit.

getLimitType()

Description

Return the type of the control limit.

Syntax

getLimitType()

- Parameters

None

- Returns

String limitType - The type of the control limit.

getLimitValue()

Description

Returns the value of the control limit.

Syntax

getLimitValue()

- Parameters

None

- Returns

String limitValue - The value of the control limit.

getResult()

Description

Returns the control limit result.

Syntax

getResult()

- Parameters

None

- Returns

boolean result - The control limit result.

Event properties:

getCp()

Description

Returns the process capability value.

Syntax

getCp()

- Parameters

None

- Returns

Double cp - The process capability value.

getCpk()

Description

Gets the process capability index.

Syntax

getCpk()

- Parameters

None

- Returns

Double cpk - The index of the process capability.

getCpl()

Description

Gets the lower process capability index.

Syntax

getCpl()

- Parameters

None

- Returns

Double cpl - The lower process capability index.

getCpm()

Description

Gets the process capability mean.

Syntax

getCpm()

- Parameters

None

- Returns

Double cpm - The process capability mean.

getCpu()

Description

Gets the upper process capability index.

Syntax

getCpu()

- Parameters

None

- Returns

Double cpu - The upper process capability index.

getCr()

Description

Gets the reciprocal of the process capability.

Syntax

getCr()

- Parameters

None

- Returns

Double cr - The reciprocal of the process capability.

getData()

Description

Gets the data corresponding to this event.

Syntax

getData()

- Parameters

None

- Returns

AnalysisDataset data - Results of the process capability event.

getLcl()

Description

Gets the lower control limit.

Syntax

getLcl()

- Parameters

None

- Returns

Double lcl - The lower control limit.

getLsl()

Description

Gets the lower specification limit.

Syntax

getLsl()

- Parameters

None

- Returns

Double lsl - The lower specification limit.

getMean()

Description

Gets the mean.

Syntax

getMean()

- Parameters

None

- Returns

Double mean - The mean.

getMeasurementCount()

Description

Gets the measurement.

Syntax

getMeasurementCount()

- Parameters

None

- Returns

int measurementCnt - The measurement count.

getStandardDeviation()

Description

Gets the estimated standard deviation.

Syntax

getStandardDeviation()

- Parameters

None

- Returns

Double standardDeviation - The estimated standard deviation.

getTarget()

Description

Gets the target.

Syntax

getTarget()

- Parameters

None

- Returns

Double target - The target.

getUcl()

Description

Gets the upper control limit.

Syntax

getUcl()

- Parameters

None

- Returns

Double ucl - The upper control limit.

getUsl()

Description

Gets the upper specification limit.

Syntax

getUsl()

- Parameters

None

- Returns

Double usl - The upper specification limit.

setCp(cp)

Description

Sets the process capability value.

Syntax

setCp(cp)

- Parameters

Double cp - The value to set as the process capability.

- Returns

Nothing

setCpk(cpk)

Description

Sets the process capability index.

Syntax

setCpk(cpk)

- Parameters

Double cpk - The process capability index to set for.

- Returns

Nothing

setCpl(cpl)

Description

Sets the lower process capability index.

Syntax

setCpl(cpl)

- Parameters

Double cpl - The lower process capability index to set for.

- Returns

Nothing

setCpm(cpm)

Description

Sets the process capability mean.

Syntax

setCpm(cpm)

- Parameters

Double cpm - The process capability mean to set for.

- Returns

Nothing

setCpu(cpu)

Description

Sets the upper process capability index.

Syntax

setCpu(cpu)

- Parameters

Double cpu - The upper process capability index to set for.

- Returns

Nothing

setCr(cr)

Description

Sets the reciprocal of the process capability.

Syntax

setCr(cr)

- Parameters

Double cr - The reciprocal of the process capability to set for.

- Returns

Nothing

setLcl(lcl)

Description

Sets the lower control limit.

Syntax

setLcl(lcl)

- Parameters

Double lcl - The lower control limit to set for.

- Returns

Nothing

setMean(mean)

Description

Sets the mean.

Syntax

setMean(mean)

- Parameters

Double mean - The mean value to set for.

- Returns

Nothing

setStandardDeviation(standardDeviation)

Description

Sets the estimated standard deviation.

Syntax

setStandardDeviation(standardDeviation)

- Parameters

Double standardDeviation - The estimated standard deviation to set for.

- Returns

Nothing

setUcl(ucl)

Description

Sets the upper control limit.

Syntax

setUcl(ucl)

- Parameters

Double ucl - The upper control limit to set for.

- Returns

Nothing

Event properties:

getCp()

Description

Returns the process capability value.

Syntax

getCp()

- Parameters

None

- Returns

Double cp - The process capability value.

getCpk()

Description

Gets the process capability index.

Syntax

getCpk()

- Parameters

None

- Returns

Double cpk - The index of the process capability.

getCpl()

Description

Gets the lower process capability index.

Syntax

getCpl()

- Parameters

None

- Returns

Double cpl - The lower process capability index.

getCpLcl()

Description

Gets the lower control limit.

Syntax

getCpLcl()

- Parameters

None

- Returns

Double cpLCL - The lower control limit.

getCpm()

Description

Gets the process capability mean.

Syntax

getCpm()

- Parameters

None

- Returns

Double cpm - The process capability mean.

getCpStandardDeviation()

Description

Gets the standard deviation.

Syntax

getCpStandardDeviation()

- Parameters

None

- Returns

Double cpStandardDeviation - The estimated standard deviation.

getCpu()

Description

Gets the upper process capability index.

Syntax

getCpu()

- Parameters

None

- Returns

Double cpu - The upper process capability index.

getCpUcl()

Description

Gets the upper control limit.

Syntax

getCpUcl()

- Parameters

None

- Returns

Double cpUCL - The upper control limit.

getCr()

Description

Gets the reciprocal of the process capability.

Syntax

getCr()

- Parameters

None

- Returns

Double cr - The reciprocal of the process capability.

getData()

Description

Gets the data corresponding to this event.

Syntax

getData()

- Parameters

None

- Returns

AnalysisDataset data - Results of the process capability event.

getLsl()

Description

Gets the lower specification limit.

Syntax

getLsl()

- Parameters

None

- Returns

Double lsl - The lower specification limit.

getMean()

Description

Gets the mean.

Syntax

getMean()

- Parameters

None

- Returns

Double mean - The mean.

getMeasurementCount()

Description

Gets the measurement.

Syntax

getMeasurementCount()

- Parameters

None

- Returns

int measurementCount - The measurement count.

getPp()

Description

Gets the process performance.

Syntax

getPp()

- Parameters

None

- Returns

Double Pp - The process performance.

getPpk()

Description

Gets the process performance index.

Syntax

getPpk()

- Parameters

None

- Returns

Double ppk - The process performance index.

getPpl()

Description

Gets the lower process performance index.

Syntax

getPpl()

- Parameters

None

- Returns

Double ppl - The lower process performance index.

getPpLcl()

Description

Gets the lower control limit of process performance.

Syntax

getPpLcl()

- Parameters

None

- Returns

Double ppLCL - The lower control limit of process performance.

getPpm()

Description

Gets the process performance index of the mean.

Syntax

getPpm()

- Parameters

None

- Returns

Double ppm - The process performance index of the mean.

getPpStandardDeviation()

Description

Gets the standard deviation.

Syntax

getPpStandardDeviation()

- Parameters

None

- Returns

Double ppStandardDeviation - The process performance standard deviation.

getPpu()

Description

Gets the upper process performance index.

Syntax

getPpu()

- Parameters

None

- Returns

Double ppu - The upper process performance index.

getPpUcl()

Description

Gets the upper control limit of process performance.

Syntax

getPpUcl()

- Parameters

None

- Returns

Double ppUCL - The upper control limit of process performance.

getPr()

Description

Gets the reciprocal of the process performance.

Syntax

getPr()

- Parameters

None

- Returns

Double pr - The reciprocal of the process performance.

getTarget()

Description

Gets the target.

Syntax

getTarget()

- Parameters

None

- Returns

Double target - The target.

getUsl()

Description

Gets the upper specification limit.

Syntax

getUsl()

- Parameters

None

- Returns

Double usl - The upper specification limit.

setCp(cp)

Description

Sets the process capability value.

Syntax

setCp(cp)

- Parameters

Double cp - The value to set as the process capability.

- Returns

Nothing

setCpk(cpk)

Description

Sets the process capability index.

Syntax

setCpk(cpk)

- Parameters

Double cpk - The process capability index to set for.

- Returns

Nothing

setCpl(cpl)

Description

Sets the lower process capability index.

Syntax

setCpl(cpl)

- Parameters

Double cpl - The lower process capability index to set for.

- Returns

Nothing

setCpLcl(cpLCL)

Description

Sets the lower control limit.

Syntax

setCpLcl(cpLCL)

- Parameters

Double cpLCL - The lower control limit to set for.

- Returns

Nothing

setCpm(cpm)

Description

Sets the process capability mean.

Syntax

setCpm(cpm)

- Parameters

Double cpm - The process capability mean to set for.

- Returns

Nothing

setCpStandardDeviation(cpStandardDeviation)

Description

Sets the standard deviation.

Syntax

setCpStandardDeviation(cpStandardDeviation)

- Parameters

Double cpStandardDeviation - The estimated standard deviation to set for.

- Returns

Nothing

setCpStandardDeviation(cpStandardDeviation)

Description

Sets the standard deviation.

Syntax

setCpStandardDeviation(cpStandardDeviation)

- Parameters

Double cpStandardDeviation - The estimated standard deviation to set for.

- Returns

Nothing

setCpu(cpu)

Description

Sets the upper process capability index.

Syntax

setCpu(cpu)

- Parameters

Double cpu - The upper process capability index to set for.

- Returns

Nothing

setCpUcl(cpUCL)

Description

Sets the upper control limit.

Syntax

setCpUcl(cpUCL)

- Parameters

Double cpUCL - The upper control limit to set for.

- Returns

Nothing

setCr(cr)

Description

Sets the reciprocal of the process capability.

Syntax

setCr(cr)

- Parameters

Double cr - The reciprocal of the process capability to set for.

- Returns

Nothing

setMean(mean)

Description

Sets the mean.

Syntax

setMean(mean)

- Parameters

Double mean - The mean value to set for.

- Returns

Nothing

setPp(pp)

Description

Sets the process performance.

Syntax

setPp(pp)

- Parameters

Double Pp - The process performance to set for.

- Returns

Nothing

setPpk(ppk)

Description

Sets the process performance index.

Syntax

setPpk(ppk)

- Parameters

Double ppk - The process performance index to set for.

- Returns

Nothing

setPpl(ppl)

Description

Sets the lower process performance index.

Syntax

setPpl(ppl)

- Parameters

Double ppl - The lower process performance index to set for.

- Returns

Nothing

setPpLcl(ppLCL)

Description

Sets the lower control limit of the process performance.

Syntax

setPpLcl(ppLCL)

- Parameters

Double ppLCL - The lower control limit to set for.

- Returns

Nothing

setPpm(ppm)

Description

Sets the process performance index of the mean.

Syntax

setPpm(ppm)

- Parameters

Double ppm - The process performance index of the mean to set for.

- Returns

Nothing

setPpStandardDeviation(ppStandardDeviation)

Description

Sets the standard deviation.

Syntax

setPpStandardDeviation(ppStandardDeviation)

- Parameters

Double ppStandardDeviation - The standard deviation to set for.

- Returns

Nothing

setPpu(ppu)

Description

Sets the upper process performance index.

Syntax

setPpu(ppu)

- Parameters

Double ppu - The upper process performance index to set for.

- Returns

Nothing

setPpUcl(ppUCL)

Description

Sets the upper control limit of the process performance.

Syntax

setPpUcl(ppUCL)

- Parameters

Double ppUCL - The upper control limit of the process performance.

- Returns

Nothing

setPr(pr)

Description

Sets the reciprocal of the process performance.

Syntax

setPr(pr)

- Parameters

Double pr - The reciprocal of the process performance to set for.

- Returns

Nothing

None

Sample Approval Updated Event

After the sample approval state has been updated, any script in this event is run. This includes samples that are set for automatic approval. It is provided to allow for the performance of other actions when sample approval state changes.

Event properties:

getSample()

Description

Returns the sample for which the approval state changed. (See Sample section more information).

Syntax

getSample()

- Parameters

None

- Returns

The sample for which the approval state changed.

isApproval()

Description

Returns true if the sample has been approved.

Syntax

isApproval()

- Parameters

None

- Returns

boolean - True if the sample has been approved.

isUnApproval()

Description

Returns true if the sample has been unapproved.

Syntax

isUnApproval()

- Parameters

None

- Returns

boolean - True if the sample has been unapproved.

Code Example

```
#Add 1 to an unrelated SQLTag value
val = system.tag.getTagValue('[Default]Quality\Test\Sample
Approval Updated')
val = val + 1
system.tag.writeToTag('[Default]Quality\Test\Sample Approval
Updated', val, 1)
```

Sample Coming Due Event

When a sample due state changes to COMING_DUE, any script in this event is run. It is provided to allow for the performance of other actions, such as alerts, when sample is coming due.

Event properties:

getSample()

Description

Returns the sample that just became due. (See Sample section for more information).

Syntax

getSample()

- Parameters

None

- Returns

Sample sample - The Sample that has just became due.

getState()

Description

Returns the current sample due state (See Sample Due State Types for more information).

Syntax

getState()

- Parameters

None

- Returns

SampleDueStateTypes The due state of this sample.

Sample Due Event

When a sample due state changes to DUE, any script in this event is run. It is provided to allow for the performance of other actions, such as alerts, when sample is due.

Event properties:

getSample()

Description

Returns the sample that just became due (See Sample section for more information).

Syntax

getSample()

- Parameters

None

- Returns

Sample sample - The sample that became due.

getState()

Description

Returns the current sample due state (See Sample Due State Types for more information).

Syntax

getState()

- Parameters

None

- Returns

SampleDueStateTypes The due state of this sample.

The following are the due state types for a sample:

UNKNOWN

COMING_DUE

DUE

OVERDUE

WAITING_APPROVAL

Sample Interval Event

The event is created to execute the auto scheduling of samples. It evaluates the sample interval and determines when to create a new sample.

Event properties:

getActiveSamples()

Description

Returns a list of samples that are currently active.

Syntax

getActiveSamples()

- Parameters

None

- Returns

List<ActiveSample> activeSamples - The list of active samples.

getComingDueMin()

Description

Returns the default coming due minutes setting for this interval event. The value represents the number of minutes required before a sample is due until the sample is considered coming due.

Syntax

getComingDueMin()

- Parameters

None

- Returns

Double comingDueMin - The coming due minute setting for this event.

getCreateSample()

Description

Checks whether a sample has been created or not.

Syntax

getCreateSample()

- Parameters

None

- Returns

Boolean True if a sample is created and False otherwise.

getDefUUID()

Description

Returns the definition UUID for the sample corresponding to this event.

Syntax

getDefUUID()

- Parameters

None

- Returns

String defUUID - The uuid definition for this sample associated with this event.

getDuration()

Description

Returns the duration for the interval in minutes.

Syntax

getDuration()

- Parameters

None

- Returns

Double duration - The time duration in minutes for this interval.

getElapsedSeconds()

Description

Gets the elapsed time in seconds for this event.

Syntax

getElapsedSeconds()

- Parameters

None

- Returns

Integer elapsedSeconds - The elapsed time for this event.

getInterval()

Description

Returns the defined interval of this sample.

Syntax

getInterval()

- Parameters

None

- Returns

Double interval - The interval defined for this sample.

getLocationPath()

Description

Gets the location path for this event.

Syntax

getLocationPath()

- Parameters

None

- Returns

String path - The location path for interval event.

getOverDueMin()

Description

Gets the duration this sample is overdue in minutes.

Syntax

getOverDueMin()

- Parameters

None

- Returns

Double overdueMin - The value represents the number of minutes required after a sample is due until the sample is considered overdue.

getPathSegment(segmentName)

Description

Gets the path of the segment associated with this event.

Syntax

getPathSegment(segmentName)

- Parameters

String segmentName - The name of the segment to return the path for. Options: Enterprise, Site, Area, Line, Cell, Cell_group, Location, Storage_zone, Storage_unit.

- Returns

String path - The path of the segment.

getProductCode()

Description

Returns the product code associated with this sample event.

Syntax

getProductCode()

- Parameters

None

- Returns

String productCode - The product code associated with the sample event.

getRefNo()

Description

Returns the reference number associated with this sample event.

Syntax

getRefNo()

- Parameters

None

- Returns

String refNo - The reference number associated with this sample event.

getRefresh()

Description

Checks whether automatic refresh is set.

Syntax

getRefresh()

- Parameters

None

- Returns

Boolean refresh - True, if automatic refresh is set and False otherwise.

getScheduleFinish()

Description

Returns the date for taking this sample is scheduled to be complete.

Syntax

getScheduleFinish()

- Parameters

None

- Returns

Date scheduleFinish - The date for the schedule to end.

getScheduleStart()

Description

Returns the date for this sample is scheduled to be taken.

Syntax

getScheduleStart()

- Parameters

None

- Returns

Date scheduledStart - The date for the sample to start.

getSecSinceLastSampleScheduled()

Description

Returns the seconds since the last sample was scheduled.

Syntax

getSecSinceLastSampleScheduled()

- Parameters

None

- Returns

Integer secSinceLastSampleScheduled - The time in seconds since the last sample was scheduled.

getSecSinceLastSampleTaken()

Description

Returns the seconds since the last sample was taken.

Syntax

getSecSinceLastSampleTaken()

- Parameters

None

- Returns

Integer secSinceLastSampleTaken - The time in seconds since the last sample was taken.

getSequenceDate()

Description

Returns the sequence date of the sample interval. Sequence date is the date representing the start of the current interval.

Syntax

getSequenceDate()

- Parameters

None

- Returns

Date sequenceDate - Start date of the current interval.

getSequenceNo()

Description

Returns the sequence number corresponding to the sample event.

Syntax

getSequenceNo()

- Parameters

None

- Returns

Integer sequenceNo - The sequence number associated with this event.

getShift()

Description

Returns the shift number.

Syntax

getShift()

- Parameters

None

- Returns

int shift - Shift for which the sample was taken.

getTag()

Description

Returns the tag associated with this sample.

Syntax

getTag()

- Parameters

None

- Returns

String tag - The tag associated with the sample.

getTraceEnabled()

Description

Checks whether the trace is enabled or not.

Syntax

getTraceEnabled()

- Parameters

None

- Returns

boolean True if trace is enabled and False otherwise.

getTraceEndedAt()

Description

Returns the date at which trace ended.

Syntax

getTraceEndedAt()

- Parameters

None

- Returns

Date traceEndedAt - The trace ended date.

getTraceStartedAt()

Description

Returns the Date at which the trace started.

Syntax

getTraceStartedAt()

- Parameters

None

- Returns

Date traceStartedAt - The date at which trace started.

getValue()

Description

Returns the value of the sample interval event.

Syntax

getValue()

- Parameters

None

- Returns

Object value - The value for this interval.

getValueChangeCount()

Description

Returns the number of time the associated value has changed.

Syntax

getValueChangeCount()

- Parameters

None

- Returns

Integer valueChangeCount - The count of the value change.

getValueChangedTimeStamp()

Description

Returns the time stamp of the value changed.

Syntax

getValueChangedTimeStamp()

- Parameters

None

- Returns

Date timeStamp - The date at which the value changed.

isShiftChangeEvent()

Description

Checks if the shift has changed.

Syntax

isShiftChangeEvent()

- Parameters

None

- Returns

Boolean shiftChangeEvent - True if the shift has changed and False otherwise.

isTracedEndedEvent()

Description

Checks whether the trace has ended.

Syntax

isTracedEndedEvent()

- Parameters

None

- Returns

Boolean True if trace ended and False otherwise.

isTracedStartedEvent()

Description

Checks whether trace has started.

Syntax

isTracedStartedEvent()

- Parameters

None

- Returns

Boolean traceStartedEvent - True if trace has started and False otherwise.

isValueChangedEvent()

Description

Checks whether the value has changed.

Syntax

isValueChangedEvent()

- Parameters

None

- Returns

Boolean valueChangedEvent - True if value has changed and False otherwise.

setCreateSample(createSample)

Description

Sets this event to create a sample.

Syntax

setCreateSample(createSample)

- Parameters

Boolean createSample - The boolean to set this property.

- Returns

Nothing

setRefresh(refresh)

Description

Sets the refresh property for this event.

Syntax

setRefresh()

- Parameters

Boolean refresh - Set it to True if the event should be automatically refreshed and False otherwise.

- Returns

Nothing

setScheduleFinish(Date)

Description

Sets the date that this event is scheduled to be completed.

Syntax

setScheduleFinish(Date)

- Parameters

Date scheduleFinish - The date for the schedule to end.

- Returns

Nothing

setScheduleStart(Date)

Description

Sets the date that this sample is scheduled to be taken.

Syntax

setScheduleStart(Date)

- Parameters

Date scheduleStart - The date for the schedule to start.

- Returns

Nothing

Sample Overdue Event

When a sample due state changes to OVERDUE, any script in this event is run. It is provided to allow for the performance other actions, such as alerts, when sample is overdue.

Event properties:

getSample()

Description

Returns the sample that is overdue (See Sample section more information).

Syntax

getSample()

- Parameters

None

- Returns

Sample sample - The Sample that is overdue.

getState()

Description

Returns the current sample due state (See Sample Due State Types for more information).

Syntax

getState()

- Parameters

None

- Returns

SampleDueStateTypes The due state of this sample.

Sample Waiting Approval Event

When a sample due state changes to WAITING_APPROVAL, any script in this event is run. It is provided to allow for the performance of other actions, such as alerts, when sample is awaiting approval.

Event properties:

getSample()

Description

Returns the sample that just became due (See Sample section for more information).

Syntax

getSample()

- Parameters

None

- Returns

Sample - The sample that just became due.

getState()

Description

Returns the current sample due state (See Sample Due State Types for more information).

Syntax

getState()

- Parameters

None

- Returns

SampleDueStateTypes - The current sample due state.

Signal Evaluated Event

When sample data changes, all of the out of control signals associated with it will be evaluated. After each attribute for each definition has been evaluated, any script in this event is run. It is provided to allow for special handling to override out of control conditions as described below. A preferred alternative is to implement the desired results in an Interval (See Intervals for more information).

Event properties:

isIgnoreOutOfControl()

Description

This script function will check if ignoreOutOfControl property is enabled or not. Whenever the signal goes out of control it is ignored if this is set to true.

Syntax

isIgnoreOutOfControl()

- Parameters

None

- Returns

boolean ignoreOutOfControl - True if the ignoreOutOfControl property of this signal event is enabled and False otherwise.

getDefUUID()

Description

Returns the definition UUID that was evaluated (See Sample Definition section more information).

Syntax

getDefUUID()

- Parameters

None

- Returns

String defUUID - The uuid that was evaluated.

setIgnoreOutOfControl(ignoreOutOfControl)

Description

Used to override and ignore an out of control condition.

Syntax

setIgnoreOutOfControl(ignoreOutOfControl)

- Parameters

boolean ignoreOutOfControl - Set it to True if an out of control condition is to be ignored and False otherwise.

- Returns

Nothing

isForceOutOfControl()

Description

This script function will check if forceOutOfControl property is enabled or not. The signal goes out of control whenever this is set to true.

Syntax

isForceOutOfControl()

- Parameters

None

- Returns

boolean forceOutOfControl - True if the forceOutOfControl property of this signal event is enabled and False otherwise.

setForceOutOfControl(forceOutOfControl)

Description

Used to force an out of control condition.

Syntax

setForceOutOfControl(forceOutOfControl)

- Parameters

boolean ignoreOutOfControl - Set it to True in order to force an out of control condition and False otherwise.

- Returns

None

getEvaluationResults()

Description

Returns a list of evaluation results. When sample data is updated for a location - sample definition combination, all of the selected signals are evaluated. This occurs for each attribute within the sample definition.

Syntax

getEvaluationResults()

- Parameters

None

- Returns

SignalEvaluationResults results - The list of results obtained during the evaluation of the signal.

getLocationPath()

Description

Gets the location path for this signal event to take place.

Syntax

getLocationPath()

- Parameters

None

- Returns

String locationPath - The location path for this signal event.

getPathSegment(segmentName)

Description

Gets the path for the specified segment.

Syntax

getPathSegment(segmentName)

- Parameters

String segmentName - Name of the segment to get the path for. Options: Enterprise, Site, Area, Line, Cell, Cell_group, Location, Storage_zone, Storage_unit.

- Returns

String pathSegment - The path segment corresponding to specified by segment name parameter.

Example:

If sample definition viscosity has an allowable location processing, has two attributes of cold viscosity and temperature, and signal rule 1 and signal rule 2 are selected, then when a sample is added or updated,cold viscosity for signal rule 1, cold viscosity for signal rule 2, temperature for signal rule 1 and temperature for signal rule 2 are all evaluated. The outcome for each combination is an item within the evaluation results returned from the getEvaluationResults() function.

Signal Evaluation Results

This object holds the evaluation results for an attribute signal combination.

Event properties:

getDefinitionName()

Description

Returns definition name of the signal which was evaluated.

Syntax

getDefinitionName()

- Parameters

None

- Returns

String definitionName - The definition name of this signal.

getSignalName()

Description

Returns the name of the signal associated with this result.

Syntax

getSignalName()

- Parameters

None

- Returns

String name - Name of the evaluated signal.

getAttributeName()

Description

Returns the name of the attribute associated with this result.

Syntax

getAttributeName()

- Parameters

None

- Returns

String attributeName - Name of the attribute associated with this result.

getViolatingSampleDate()

Description

Returns the date of the most recent sample that is in violation of the signal.

Syntax

getViolatingSampleDate()

- Parameters

None

- Returns

Date violatingSampleDate - The date of sample to which the violation of signal has occurred recently.

setViolatingSampleDate(violatingSampleDate)

Description

Set the date of the most recent sample that is in violation of the signal.

Syntax

setViolatingSampleDate(violatingSampleDate)

- Parameters

Date v iolatingSampleDate - The date of the most recent sample that is in violation of the signal.

- Returns

Nothing

setLastSampleDate(lastSampleDate)

Description

Set the date for the last approved sample.

Syntax

setLastSampleDate(lastSampleDate)

- Parameters

Date lastSampleDate - The date to be set for the sample which was recently approved.

- Returns

Nothing

getLastSampleDate()

Description

Returns the date of the last approved sample. This can be used in combination to determine if the last approved sample caused the signal violation.

Syntax

getLastSampleDate()

- Parameters

None

- Returns

Date lastSampleDate - The date to be set for the sample which was recently approved.

isSignalViolation()

Description

Returns true if the signal - attribute combination are in violation.

Syntax

isSignalViolation()

- Parameters

None

- Returns

boolean True if the signal is violated and False otherwise.

setEvaluationError(message)

Description

Set the message for any evaluation error.

Syntax

setEvaluationError(message)

- Parameters

String message - The note to be set for any evaluation error.

- Returns

None

isEvaluationError()

Description

Returns true if an error occurred during the signal evaluation.

Syntax

isEvaluationError()

- Parameters

None

- Returns

boolean True if an error occurred during the signal evaluation and False otherwise.

hasMessage()

Description

Returns true if a message exists.

Syntax

hasMessage()

- Parameters

None

- Returns

boolean - True if a message exists and False otherwise.

getMessage()

Description

Returns textual description of error encountered during the signal evaluation.

Syntax

getMessage()

- Parameters

None

- Returns

String message - The description of error encountered during the signal evaluation.

setMessage(message)

Description

Set the textual description of error encountered during the signal evaluation.

Syntax

setMessage(message)

- Parameters

String message - The note to be displayed when an error occurs.

- Returns

None

getLocationPath()

Description

Gets the location path for this evaluated signal.

Syntax

getLocationPath()

- Parameters

None

- Returns

String locationPath - The location path for this signal.

getPathSegment(segmentName)

Description

Gets the path for the specified segment.

Syntax

getPathSegment(segmentName)

- Parameters

String segmentName - Name of the segment to get the path for. Options: Enterprise, Site, Area, Line, Cell, Cell_group, Location, Storage_zone, Storage_unit.

- Returns

String pathSegment - The path segment corresponding to specified by segment name parameter.

Signal In Control Event

When sample data changes, all of the out of control signals associated with it will be evaluated. If an out of control signal changes from **Out of Control** to **In Control**, any script in this event is run. It is provided to allow for the performance of other actions, such as alerts, when an out of control condition no longer exists.

Event properties:

getDefUUID()

Description

Returns the definition UUID associated with this in control event (See Sample Definition section more information).

Syntax

getDefUUID()

- Parameters

None

- Returns

String uuid - The definition uuid associated with this event.

getEvaluationResults()

Description

Returns a single evaluation result of the signal - attribute combination that transitioned from out of control to in control.

Syntax

getDefUUID()

- Parameters

None

- Returns

SignalEvaluationResults - The results of the evaluation.

Signal Out of Control Event

When sample data changes, all of the out of control signals associated with it will be evaluated. If an out of control signal changes from **In Control** to **Out of Control**, any script in this event is run. It is provided to allow for the performance of other actions, such as alerts, when an out of control condition occurs.

Event properties:

getDefUUID()

Description

Returns the definition UUID associated with this out of control event (See Sample Definition section more information).

Syntax

getDefUUID()

- Parameters

None

- Returns

String uuid - The definition uuid associated with this event.

getEvaluationResults()

Description

Returns a single evaluation result of the signal - attribute combination that transitioned from in control to out of control.

Syntax

getEvaluationResults()

- Parameters

None

- Returns

SignalEvaluationResults - Result of signal evaluation.

9.6.6 Recipe Objects

The Sepasoft Recipe / Changeover Module exposes many script functions that support managing recipes. In fact, the internal functions used by the recipe editor and other recipe components are exposed as script functions that can be used on the client or the gateway. The Recipe Module has script functions and events that use various objects. This is because some recipe information contains more data that can be represented with a single primitive data type. For example a recipe value has a name, description, units, format and more, and the Item Recipe Value is used to hold all of this information when returning back recipe value information from a script function. When recipe values are added to a production item, there are properties that allow script to be entered. The following sections detail the different events and how they are used.

Change Log Filters

A ChangeLogFilters object is used when requesting recipe change logs with the system.recipe.getChangelogHistory script function to narrow down the results that are returned. For example, if you only want the change log history for a specific production item (machine) and specific date range, the ChangeLogFilters object properties are set appropriately and are passed as parameters to the system.recipe.getChangelogHistory script function.

Methods:

createNew()

Description

Returns a new instance of a ChangeLogFilters object.

Info

After setting various filter properties, it is used with the system.recipe.getChangelogHistory script function.

Syntax

createNew()

- Parameters

None

- Returns

The new ChangeLogFilters object.

- Scope

All

Properties:

addCategory(category)

Description

Add a category to include in the recipe change log history results.

Info

Multiple categories can be specified to be included in the results.

Valid values are :

- RECIPE to include change log entries dealing with recipe changes. Recipe changes include, adding new recipes, renaming recipes, deleting recipes, adding production items to recipes, etc.
- RECIPE_VALUE to include change log entries dealing with changes of recipe values. This includes changing a value, reverting a value back to be inherited, etc.
- SUB_PRODUCT_CODE to include change log entries dealing with sub product codes. This includes adding new sub product codes, renaming sub product codes, deleting sub product codes, etc.
- SUB_PRODUCT_CODE_VALUE to include change log entries dealing with changes of sub product code values or default values for a production item. This includes changing a value, reverting a value back to be inherited, etc.

Syntax

addCategory(category)

- Parameters

String category - The category to include in the change log history for.

- Returns

Nothing

- Scope

All

removeCategory(category)

Description

Remove a category for what has already been added.

Info

Multiple categories can be specified to be removed.

Valid values are :

- RECIPE to include change log entries dealing with recipe changes. Recipe changes include, adding new recipes, renaming recipes, deleting recipes, adding production items to recipes, etc.
- RECIPE_VALUE to include change log entries dealing with changes of recipe values. This includes changing a value, reverting a value back to be inherited, etc.
- SUB_PRODUCT_CODE to include change log entries dealing with sub product codes. This includes adding new sub product codes, renaming sub product codes, deleting sub product codes, etc.
- SUB_PRODUCT_CODE_VALUE to include change log entries dealing with changes of sub product code values or default values for a production item. This includes changing a value, reverting a value back to be inherited, etc.

Syntax

removeCategory(category)

- Parameters

String category - The category to be removed from the log history for.

- Returns

Nothing

- Scope

All

setFromDate(fromDate)

Description

Set the start of the date range to return change log history for.

Syntax

setFromDate(fromDate)

- Parameters

Date fromDate - The start date to return the change log history for.

- Returns

Nothing

- Scope

All

setItemPathFilter(itemPath)

Description

Set the path of the production item to return change log history for.

Syntax

setItemPathFilter(itemPath)

- Parameters

String itemPath - The item path to a production line, cell, cell group or location.

- Returns

Nothing

- Scope

All

setProjectName(projectName)

Description

Set the project name to read recipe change log history.

Info

Recipe change log history is kept by project, and the project name is required with the system.recipe.getChangelogHistory script function.

Syntax

setProjectName(projectName)

- Parameters

String projectName - The project name to read the change log history for.

- Returns

Nothing

- Scope

All

setRecipeNameFilter(recipeNameFilter)

Description

Set an optional recipe filter.

Info

The filter can contain ? and * wild card characters. For example: "Recipe C*" will include all recipes that start with Recipe C. Recipe C1 and Recipe C21 will be included but Recipe D1 will not.

Syntax

setRecipeNameFilter(recipeNameFilter)

- Parameters

String recipeNameFilter - The recipe name to filter the results for.

- Returns

Nothing

- Scope

All

setSubProductCodeFilter(subProductCodeFilter)

Description

Set an optional sub product code filter. The filter can contain ? and * wild card characters.

Syntax

setSubProductCodeFilter(subProductCodeFilter)

- Parameters

String subProductCodeFilter - The subProductCode to filter the results for.

- Returns

Nothing

- Scope

All

setToDate(toDate)

Description

Set the end of the date range to return change log history for.

Syntax

setToDate(toDate)

- Parameters

Date toDate - The end date to return the change log history for.

- Returns

Nothing

- Scope

All

setUserFilter(userNameFilter)

Description

Set an optional user name filter. The filter can contain ? and * wild card characters.

Syntax

setUserFilter(userNameFilter)

- Parameters

String userNameFilter - The userName to filter the results for.

- Returns

Nothing

- Scope

All

setValueNameFilter(recipeValueNameFilter)

Description

Set an optional recipe value name filter. The filter can contain ? and * wild card characters.

Syntax

setValueNameFilter(recipeValueNameFilter)

- Parameters

String recipeValueNameFilter - The recipeValueName to filter the results for.

- Returns

Nothing

- Scope

All

Code Snippets

```
#Collect values we want to filter by
projectName = system.util.getProjectName()
itemPath = event.source.parent.getComponent('Production Line
Selector').selectedPathWithoutProject
fromDate = event.source.parent.getComponent('Date Range').
startDate
toDate = event.source.parent.getComponent('Date Range').endDate

#Build the filters object
filters = system.recipe.filter.changelog.createNew()
filters.setProjectName(projectName)
filters.addCategory("Recipe")
filters.setItemPathFilter(itemPath)
filters.setFromDate(fromDate)
```

```
filters.setToDate(toDate)

#Request the change log for the given filters
ds = system.recipe.getChangelogHistory(filters)
event.source.parent.getComponent('Table').data = ds
```

Evaluate Variance Script

If the Evaluate Variance Script property (see Enable Variance Monitoringfor more information) of a production item contains a script, it will be executed every time an Ignition tag associated with a recipe value changes. Variance monitoring must also be active. When it is executed, the event object provides information about the recipe value being evaluated. If event. setLogVariance(logVariance) is not called, then the log variance state determined before this event will be used. See Variance Monitoring for more information.

Properties:

getRecipeTag()

Description

Get the tag associated with the recipe value being evaluated.

Syntax

getRecipeTag ()

- Parameters

None

- Returns

RecipeTag tag - An instance of a RecipeTag.

getScale()

Description

Return the scale factor for the MES production item associated with the recipe value being evaluated.

Syntax

getScale ()

- Parameters

None

- Returns

Double scale - The scale factor associated with the evaluated recipe value.

isLogVariance()

Description

Get the is log variance flag for the recipe value being evaluated.

Syntax

isLogVariance ()

- Parameters

None

- Returns

boolean logVariance - True is returned if recipe value being evaluated is outside of the configured variance settings and False otherwise.

setLogVariance()

Description

Set the variance flag for the recipe value being evaluated. Use to override the current state of the variance flag.

Syntax

setLogVariance ()

- Parameters

boolean logVariance - True to flag recipe value as in variance, else return false.

- Returns

Nothing

Code Example

```
upperLimit = system.tag.read("[Default]UpperLimit")
lowerLimit = system.tag.read("[Default]LowerLimit")
recipeValue = event.getRecipeTag().getCurrentValue()
if recipeValue > upperLimit.value or recipeValue < lowerLimit.
value:
 event.setLogVariance(True)
else:
 event.setLogVariance(False)
```

Item Recipe Value

A ItemRecipeValue object is returned by many of the recipe script methods, usually as a list of ItemRecipeValue. Because a recipe value has several properties such as the name, minimum value, maximum value, units, etc., the details are returned in this ItemRecipeValue object.

Recipe values also can have varying data types and an ItemRecipeValue object supports reading the value in its true data type. For example, if a recipe value if of type Float8, then getValue(), getMinValue() and getMaxValue() all return Float8 values. This cannot be done by returning the recipe values in a Dataset where each column must be a single data type.

Properties:

getAssignedBy()

Description

Returns the recipe or production item default value where the recipe value is inherited from. If the recipe value is not inherited, it will be the name of the recipe where it was overridden.

Syntax

getAssignedBy()

- Parameters

None

- Returns

The default value of the item.

- Scope

All

getCategory()

Description

Returns the MES module that created the recipe value. Where 1 is recipe value created by the recipe module, 2 is recipe value created by the OEE module and 3 is recipe value created by the SPC module.

Syntax

getCategory()

- Parameters

None

- Returns

The MES module that created the recipe.

- Scope

All

getDataType()

Description

Returns the Ignition DataType for the recipe value.

Syntax

getDataType()

- Parameters

None

- Returns

DataType for the specified recipe value.

- Scope

All

getDescription()

Description

Returns the description of the recipe value. This is the description entered in the recipe value entry in the designer.

Syntax

getDescription()

- Parameters

None

- Returns

The description of the recipe value.

- Scope

All

getFormat()

Description

Returns the format as defined in the Ignition tag.

Syntax

getFormat()

- Parameters

None

- Returns

The format of the recipe value.

- Scope

All

getMinValue()

Description

Returns the minimum value that the value can be as defined in the Ignition tag. This is different from the recipe value security that depends on the authenticated user. See Recipe Security for more information.

Syntax

getMinValue()

- Parameters

None

- Returns

The minimum value of the recipe value.

- Scope

All

getMaxValue()

Description

Returns the maximum value that the value can be as defined in the Ignition tag. This is different from the recipe value security that depends on the authenticated user. See Recipe Security for more information.

Syntax

getMaxValue()

- Parameters

None

- Returns

The maximum value of the recipe value.

- Scope

All

getName()

Description

Returns the name of the recipe value. This is the same name entered in the recipe value entry in the designer.

Syntax

getName()

- Parameters

None

- Returns

Name of the recipe value.

- Scope

All

getSortOrder()

Description

Gets the sort order for the item recipe value.

Syntax

getSortOrder()

- Parameters

None

- Returns

Integer sortOrder - The sort order for the item recipe value.

- Scope

All

getUnits()

Description

Returns the units as defined in the Ignition tag.

Syntax

getUnits()

- Parameters

None

- Returns

The units defined for the recipe value.

- Scope

All

getValue()

Description

Returns the value of the recipe value. The data type will be one defined by the Ignition DataType. If a value has not been assigned to the recipe value, then None will be returned and isValid() will return false.

Syntax

getValue()

- Parameters

None

- Returns

The value of the recipe value.

- Scope

All

hasDataType()

Description

Returns True if the recipe value has a data type assigned. Recipe values that do not have a tag assigned to them will not have a data type. This is because the data type is obtained from the tag.

Syntax

hasDataType()

- Parameters

None

- Returns

True, if datatype is assigned.

- Scope

All

hasDescription()

Description

Returns True if a description exists for the recipe value.

Syntax

hasDescription()

- Parameters

None

- Returns

True, if the description exist.

- Scope

All

hasSortOrder()

Description

Checks whether the item recipe value has a sort order or not.

Syntax

hasSortOrder()

- Parameters

None

- Returns

boolean sortOrder - True, if there exist a sort order and False otherwise.

- Scope

All

hasValue()

Description

Checks whether there exist a valid value associated with the recipe item.

Syntax

hasValue()

- Parameters

None

- Returns

boolean True, if the value is valid and False otherwise.

- Scope

All

isValid()

Description

Returns True if the value is valid for the data type of the recipe value.

Syntax

isValid()

- Parameters

None

- Returns

True if the value is valid for the data type of the recipe value.

- Scope

All

restorePreviousValue()

Description

This script function will reset the previous value for the item recipe value.

Syntax

restorePreviousValue()

- Parameters

None

- Returns

Nothing

- Scope

All

setDataType(dataType)

Description

Convert the value to the new data type.

Syntax

setDataType(dataType)

- Parameters

DataType dataType - The new data type to be set.

- Returns

Nothing

- Scope

All

setSortOrder(sortOrder)

Description

Sets the sorting order for this recipe item.

Syntax

setSortOrder(sortOrder)

- Parameters

Integer sortOrder - The sort order to set for.

- Returns

Nothing

- Scope

All

setValue(value)

Description

Sets the value to this item recipe value.

Syntax

setValue(value)

- Parameters

Object value - The value to be set for.

- Returns

Nothing

- Scope

All

updateValue(value, assignedBy)

Description

Update the new value and assignedBy parameters for the item recipe.

Syntax

updateValue(value, assignedBy)

- Parameters

Object value - The value to be updated.

String assignedBy - The new assignedBy to get updated.

- Returns

Nothing

- Scope

All

Code Snippets

```
itemPath = event.source.parent.getComponent('Production Line
Selector').selectedPathWithoutProject
valueList = system.recipe.getDefaultValueItems(itemPath, "")
if valueList != None:
 system.gui.messageBox("Count = %d" % valueList.size())
 for itemRecipeValue in valueList:
         system.gui.messageBox("%s = %s" % (itemRecipeValue.
getName(), itemRecipeValue.getValue()))
```

Recipe Event

This event is created to get a recipe information.

Properties:

getItemPath()

Description

Get the MES production item path that the recipe is being selected or canceled.

Syntax

getItemPath()

- Parameters

None

- Returns

String itemPath - MES production item path.

- Scope

All

getRecipeName()

Description

Get the recipe name that is being selected or canceled.

Syntax

getRecipeName()

- Parameters

None

- Returns

Name of the recipe.

- Scope

All

getScale()

Description

Return the scale factor for the MES production item associated with this recipe event.

Syntax

getScale()

- Parameters

None

- Returns

Double scale - The scale factor associated with the recipe that is selected or cancelled.

- Scope

All

getTrackingUUID()

Description

Gets the tracking uuid associated with this recipe event.

Syntax

getTrackingUUID()

- Parameters

None

- Returns

String trackingUUID - The uuid meant for tracking this recipe event.

- Scope

All

isRecipeActive()

Description

This script function checks whether the selected recipe is active.

Syntax

isRecipeActive()

- Parameters

None

- Returns

boolean recipeActive - True if the selected recipe is active and False otherwise.

- Scope

All

isWriteError()

Description

When the recipe is selected, the recipe values are written to tags. If this is a success, then isWriteError property is True.

Syntax

isWriteError()

- Parameters

None

- Returns

boolean writeError - If there is an error in writing the error then isWriteError is True and False otherwise.

- Scope

All

setRecipeName(recipeName)

Description

Set the recipe name to select. Use to override the recipe being selected.

Syntax

setRecipeName(recipeName)

- Parameters

String name - Name of the recipe.

- Returns

Nothing

- Scope

All

Recipe Production Item Info

isSelectable()

Description

Returns True, if the item is selectable and False otherwise.

Syntax

isSelectable()

- Parameters

None

- Returns

boolean - True, if the recipe item is selectable and False otherwise.

- Scope

All

getItemPath()

Description

Gets the item path of this recipe production item.

Syntax

getItemPath()

- Parameters

None

- Returns

String itemPath - The item path associated with this production item.

- Scope

All

getLevel()

Description

Gets the level of this production item.

Syntax

getLevel()

- Parameters

None

- Returns

int level - The corresponding level of this production item.

- Scope

All

isSelected()

Description

Checks whether this production item is selected or not.

Syntax

isSelected()

- Parameters

None

- Returns

boolean selected - True, if this production item is selected and False otherwise.

- Scope

All

setSelected(selected)

Description

Sets this recipe production item as selected.

Syntax

setSelected(selected)

- Parameters

boolean selected - True, if you want to select this production item and False otherwise.

- Returns

Nothing

- Scope

All

isInherited()

Description

Checks if the production item is inherited.

Syntax

isInherited()

- Parameters

None

- Returns

boolean inherited - True, if this is an inherited production item and False otherwise.

- Scope

All

flagAsSelected()

Description

Flags the recipe production item as selected.

Syntax

flagAsSelected()

- Parameters

None

- Returns

Nothing

- Scope

All

flagAsInherited()

Description

Flags the recipe production item as inherited.

Syntax

flagAsInherited()

- Parameters

None

- Returns

Nothing

- Scope

All

hasSelectionChanged()

Description

Checks if the originally selected production item is different from the current selection.

Syntax

hasSelectionChanged()

- Parameters

None

- Returns

boolean selected - True, if the selection is changed and False otherwise.

- Scope

All

Recipe Tag

A RecipeTag object contains details about a recipe value. It reflects the properties that are configured in the designer when the recipe value was added plus some live information such as the current value. See Recipe Values Settings for more information.

Properties:

convertAndScaleValue(value, scaleFactor)

Description

Returns the value passed in the parameter in the correct data type for the recipe value with scaling.

Info

For more information, see Recipe Scaling and Recipe Values.

Syntax

convertValue(value, scaleFactor)

- Parameters

String value - The actual value to be scaled.

Double scaleFactor - The factor you have to multiply for each serving. Say you have a 10-serving (original number of servings) recipe that you want to scale down for six (desired number of servings) dinner guests. That's 6 ÷ 10 or .6. Your conversion factor is 6. Simply multiply each ingredient by .6 to get the exact amount for the recipe. Do the same to scale up. For 12 servings of your 10-serving recipe, divide 12 by 10 to get a conversion factor of 1.2.

- Returns

The converted value with the specified data type and scaling.

convertValue(value)

Description

Returns the value passed in the parameter in the correct data type for the recipe value. If the recipe value is an Int4, then the string value passed in the value parameter will be converted to an Int4 data type and returned.

Info

For more information, see Recipe Scaling and Recipe Values.

Syntax

convertValue(value)

- Parameters

String value -The actual value of the recipe.

- Returns

The converted value with the specified data type.

getCurrentValue()

Description

Returns the tag value of a tag associated with a recipe value.

Syntax

getCurrentValue()

- Parameters

None

- Returns

The value associated with the specified recipe value.

getDataType()

Description

Returns the data type of the tag associated with the recipe value.

Syntax

getDataType()

- Parameters

None

- Returns

The data type of recipe value tag.

getHighVarianceThresholdStatement()

Description

Returns the high variance threshold statement that was entered for the recipe value entry.

Syntax

getHighVarianceThresholdStatement()

- Parameters

None

- Returns

The high variance threshold statement for the specified recipe value.

getHighVarianceThresholdValue()

Description

Returns the high variance threshold value. This is calculated after the tag value change is detected and is used for default handling for the log variance state. It is provided here as a convenience.

Syntax

getHighVarianceThresholdValue()

- Parameters

None

- Returns

The high variance threshold value for the specified recipe value.

getLowVarianceThresholdStatement()

Description

Returns the low variance threshold statement that was entered for the recipe value entry.

Syntax

getLowVarianceThresholdStatement()

- Parameters

None

- Returns

The low variance threshold statement for the specified recipe value.

getLowVarianceThresholdValue()

Description

Returns the low variance threshold value. This is calculated after the tag value change is detected and is used for default handling for the log variance state. It is provided here as a convenience.

Syntax

getLowVarianceThresholdValue()

- Parameters

None

- Returns

The low variance threshold value for the specified recipe value.

getPreviousValue()

Description

Returns the previous value of the recipe value. Anytime the value of a tag associated with a recipe value changes, the previous value is saved internally. This is used for the changed from information in variance logging.

Syntax

getPreviousValue()

- Parameters

None

- Returns

The previously recorded value.

getRecipeValue()

Description

Returns the recipe value. This can be the value that was entered in recipe editor, set using script or inherited from a parent.

Syntax

getRecipeValue()

- Parameters

None

- Returns

The recipe value.

getRecipeValue(scale)

Description

Returns the recipe value adjusted by the scale parameter.

Info

For more information, see Recipe Scaling and Recipe Values.

Syntax

getRecipeValue(scale)

- Parameters

Double scale - The factor to be multiplied with the recipe value.

- Returns

The recipe value after scaling.

getRecipeValueName()

Description

Returns the name of the recipe value. This is the same name entered in the recipe value entry in the designer.

Syntax

getRecipeValueName()

- Parameters

None

- Returns

The name of the recipe value.

getTagPath()

Description

Returns the Ignition tag path assigned to the recipe value. This is the tag path entered for the recipe value.

Syntax

getTagPath()

- Parameters

None

- Returns

The tag path of the recipe value.

hasCurrentValue()

Description

Returns True if the recipe value has a value.

Syntax

hasCurrentValue()

- Parameters

None

- Returns

True, if the recipe value has a value.

hasHighVarianceThresholdStatement()

Description

Returns True if a high variance threshold statement was entered for the recipe value.

Syntax

hasHighVarianceThresholdStatement()

- Parameters

None

- Returns

True, if there exist a high variance threshold statement for the specified recipe value.

hasLowVarianceThresholdStatement()

Description

Returns True if a low variance threshold statement was entered for the recipe value.

Syntax

hasLowVarianceThresholdStatement()

- Parameters

None

- Returns

The low variance threshold value for the specified recipe value.

hasPreviousValue()

Description

Returns True if a previous value has been recorded for the recipe value.

Syntax

hasPreviousValue()

- Parameters

None

- Returns

True, if there exist a previously recorded value.

isVarianceMonitorEnabled()

Description

Returns True if variance monitoring is enabled for the recipe value.

Syntax

isVarianceMonitorEnabled()

- Parameters

None

- Returns

True, if variance monitoring is enabled.

scaleValue(value, scaleFactor)

Description

Scales and returns the value passed in the parameter in the same data type as the value parameter.

Info

For more information, see Recipe Scaling and Recipe Values.

Syntax

scaleValue(value, scaleFactor)

- Parameters

String value - The actual value to be scaled.

Double scaleFactor - The factor you have to multiply for each serving. Say you have a 10-serving (original number of servings) recipe that you want to scale down for six (desired number of servings) dinner guests. That's 6 ÷ 10 or .6. Your conversion factor is 6. Simply multiply each ingredient by .6 to get the exact amount for the recipe. Do the same to scale up. For 12 servings of your 10-serving recipe, divide 12 by 10 to get a conversion factor of 1.2.

- Returns

The value with the same data type as the value parameter.

Code Snippets

```
upperLimit = system.tag.read("[Default]UpperLimit")
lowerLimit = system.tag.read("[Default]LowerLimit")
recipeValue = event.getRecipeTag().getCurrentValue()

rt = event.getRecipeTag()
uLimit = rt.scaleValue(upperLimit.value, event.getScale())
lLimit = rt.scaleValue(lowerLimit.value, event.getScale())

if recipeValue > uLimit or recipeValue < lLimit:
 event.setLogVariance(True)
else:
 event.setLogVariance(False)
```

Recipe Value Security Info

A RecipeValueSecurityInfo object contains the list of security roles for a recipe value.

Properties:

addSecurityRole(securityRole, allowEdit, minValue, maxValue)

Description

This script function will add a new recipe value security role to the recipe item.

Syntax

addSecurityRole(securityRole, allowEdit, minValue, maxValue)

- Parameters

String securityRole - The security role to be added.

boolean allowEdit - Set it to True if editing should be allowed and False otherwise.

Object minValue - The minimum value of the range.

Object maxValue - The maximum value of the range.

- Returns

Nothing

checkWithinRange(value, min, max)

Description

Checks if the given value is within the minimum and maximum value.

Syntax

checkWithinRange(value, min, max)

- Parameters

Object value - The value to be checked for.

Object min - The minimum value of the range.

Object max - The maximum value of the range.

- Returns

boolean True if the value is between minimum and maximum value and False otherwise.

convertValue(value)

Description

Converts the value of the recipe item to the new value.

Syntax

convertValue(value)

- Parameters

String value - The value to set the recipe for.

- Returns

Object itemRecipeValue - The recipe item with the new value.

getAssignedBy()

Description

Returns the description of where this recipe value was assigned by.

Syntax

getAssignedBy()

- Parameters

None

- Returns

Description corresponding to the recipe value.

getItemPath()

Description

Returns the item path of the recipe value.

Syntax

getItemPath()

- Parameters

None

- Returns

The item path to a production line, cell, cell group or location corresponding to the recipe value.

getItemRecipeValue()

Description

Returns a ItemRecipeValue object.

Syntax

getItemRecipeValue()

- Parameters

None

- Returns

The Item recipe value object.

getMin(value1, value2)

Description

Get the object containing the minimum value.

Syntax

getMin(value1, value2)

- Parameters

Object value1 - The first object with value 1.

Object value2 - The second object with value 2.

- Returns

The object with minimum value.

getMax(value1, value2)

Description

Get the object containing the maximum value.

Syntax

getMax(value1, value2)

- Parameters

Object value1 - The first object with value 1.

Object value2 - The second object with value 2.

- Returns

The object with maximum value.

getSecurityRole(roleName)

Description

Returns a RecipeValueSecurityRole object.

Syntax

getSecurityRole(roleName)

- Parameters

String roleName - Role to be assigned to the security object.

- Returns

The RecipeValueSecurityRole object.

isInherit()

Description

Checks if the recipe value is inherited or not.

Syntax

isInherit()

- Parameters

None

- Returns

boolean True, if the recipe value is inherited and False otherwise.

removeSecurityRole(roleName)

Description

Removes the security role with the given role name.

Syntax

removeSecurityRole(roleName)

- Parameters

String roleName - The role name for the security role to be removed for.

- Returns

Nothing

setAssignedBy(assignedBy)

Description

Sets the assignedBy property to the recipe item.

Syntax

setAssignedBy(assignedBy)

- Parameters

String assignedBy - The assignedBy property value for the recipe item.

- Returns

Nothing

setInherit(inherit)

Description

Sets the inherit property for the recipe item.

Syntax

setInherit(inherit)

- Parameters

boolean inherit - Set to True if the recipe item should inherit the security from the parent and False otherwise.

- Returns

Nothing

validateValue(value)

Description

Validate the given item recipe value.

Syntax

validateValue(value)

- Parameters

String value - The value to check the validation for.

- Returns

String value - The validated value of this recipe item.

Recipe Value Security Role

A RecipeValueSecurityRole object contains the security information for a security role for a recipe value.

Properties:

getMaxValue()

Description

Returns the maximum value for this role.

Syntax

getMaxValue()

- Parameters

None

- Returns

The maximum value of the role.

- Scope

All

getMinValue()

Description

Returns the minimum value for this role.

Syntax

getMinValue()

- Parameters

None

- Returns

The minimum value of the role.

- Scope

All

isAdministratorRole()

Description

Returns true if this role is 'Administrator'.

Syntax

isAdministratorRole()

- Parameters

None

- Returns

True, if this role is 'Administrator'.

- Scope

All

isAllowEdit()

Description

Returns true if this value is editable.

Syntax

isAllowEdit()

- Parameters

None

- Returns

True, if the value is editable.

- Scope

All

isModified()

Description

Checks whether the security role is modified or not.

Syntax

isModified()

- Parameters

None

- Returns

boolean modified - True, if the security role is been modified and False otherwise.

- Scope

All

restorePreviousValues()

Description

This script function will restore the previous values associated with this recipe security role.

Syntax

restorePreviousValues()

- Parameters

None

- Returns

Nothing

- Scope

All

setAllowEdit(allowEdit)

Description

Sets the allow edit property for this security role.

Syntax

setAllowEdit(allowEdit)

- Parameters

boolean allowEdit - Set to True inorder to allow editing of this security role and False otherwise.

- Returns

Nothing

- Scope

All

setMinValue(minValue)

Description

Sets the minimum value for this security role.

Syntax

setMinValue(minValue)

- Parameters

Object minValue - The minimum value to set the security role for.

- Returns

Nothing

- Scope

All

setMaxValue(maxValue)

Description

Sets the maximum value for this security role.

Syntax

setMaxValue(maxValue)

- Parameters

Object maxValue - The maximum value to set the security role for.

- Returns

Nothing

- Scope

All

Request Value Script

If the Request Value Script property (see Adding a Recipe Value for more information) of a production item contains a script, it will be executed every time a recipe is selected. When it is executed, the event object provides information about the recipe value being read. If event. setRecipeValue(value) is not called, then the value from the database will be used.

Properties:

getItemPath()

Description

Return the item path for the recipe value being read.

Syntax

getItemPath ()

- Parameters

None

- Returns

String itemPath - Item path for MES production item.

getRecipeValue()

Description

Return the current value for the recipe value being read.

Syntax

getRecipeValue ()

- Parameters

None

- Returns

String recipeValue - The value for the recipe value being read.

getScale()

Description

Return the scale factor for the MES production item associated with the recipe value being read.

Syntax

getScale ()

- Parameters

None

- Returns

Double scale - The scale factor corresponding to this recipe value.

getTagPath()

Description

Return the Ignition tag path assigned to the recipe value being read.

Syntax

getTagPath ()

- Parameters

None

- Returns

String tagPath - The tag path corresponding to this recipe value.

getValueName()

Description

Return the name for the recipe value being read.

Syntax

getValueName ()

- Parameters

None

- Returns

String valueName - Name of the recipe value.

setRecipeValue(value)

Description

Set the recipe value. Used to override the value defined in the recipe for the recipe value being read.

Syntax

setRecipeValue (value)

- Parameters

String recipeValue - The new recipe value.

- Returns

Nothing

Code Example

```
import math
try:
 value = float(event.getRecipeValue())
 event.setRecipeValue(str(math.log10(value)))
except:
 event.setRecipeValue("0")
```

Variance Filters

A VarianceFilters object is used when requesting variances with the system.recipe.getRecipeVariances script function to narrow down the results that are returned. For example, if you only want variances for a specific production item (machine) and specific date range, the VarianceFilters object properties are set appropriately and are passed as parameters to the system.recipe.getRecipeVariances script function.

Methods:

createNew()

Description

Returns a new instance of a VarianceFilters object. After setting various filter properties, it is used with the system.recipe.getRecipeVariances script function.

Syntax

createNew()

- Parameters

None

- Returns

The new VarianceFilter object.

Properties:

setFromDate(fromDate)

Description

Set the start of the date range to return variances for if setVarianceScopeTypes ("DATE_RANGE") is called.

Syntax

setFromDate(fromDate)

- Parameters

Date fromDate - The start date for the variance.

- Returns

Nothing

setIncludeChildren(includeChildren)

Description

Set if children production items under the production item specified by the setItemPath() property, should be included in the variance results.

Syntax

setIncludeChildren(includeChildren)

- Parameters

Boolean includeChildren - Set it True, if you need to include children and False otherwise.

- Returns

Nothing

setIncludeInitialValues(includeInitialValues)

Description

Set if the initial values (meaning the values when the recipe was first selected) should be included in the variance results.

Syntax

setIncludeInitialValues(includeInitialValues)

- Parameters

Boolean includeInitialValues - Set it True, if you need to include the initial values and False otherwise.

- Returns

Nothing

setIncludeVarianceValues(includeVarianceValues)

Description

Set if the variance values (meaning the values that changed after the recipe was first selected) should be included in the variance results.

Syntax

setIncludeVarianceValues(includeVarianceValues)

- Parameters

Boolean includeVarianceValues - Set it True, if you need to include the variance values and False otherwise.

- Returns

Nothing

setItemPath(itemPath)

Description

Set the path of the production item to return variances for.

Syntax

setItemPath(itemPath)

- Parameters

String itemPath - The item path to a production line, cell, cell group or location.

- Returns

Nothing

setProjectName(projectName)

Description

Set the project name to read variances. Variances are kept by project, and the project name is required with the getRecipeVariances script function.

Syntax

setProjectName(projectName)

- Parameters

String projectName - The project name to read the variances for.

- Returns

Nothing

setRecipe(recipeName)

Description

Set an optional recipe filter. The filter can contain ? and * wild card characters. For example: "Recipe C*" will include all recipes that start with Recipe C. Recipe C1 and Recipe C21 will be included but Recipe D1 will not.

Syntax

setRecipe(recipeName)

- Parameters

String recipeName - Name of the recipe value. This is the same name entered in the recipe value entry in the designer.

- Returns

Nothing

setRecipeValueName(recipeValueName)

Description

Set an optional recipe value name filter. The filter can contain ? and * wild card characters.

Syntax

setRecipeValueName(recipeValueName)

- Parameters

String recipeValueName - N ame of the recipe value. This is the same name entered in the recipe value entry in the designer.

- Returns

Nothing

setSubRecipe(subRecipeName)

Description

Set an optional sub recipe filter. The filter can contain ? and * wild card characters. See Sub Recipes for more information.

Syntax

setSubRecipe(subRecipeName)

- Parameters

String subRecipeName - N ame of the subRecipe value. This is the same name entered in the subRecipe value entry in the designer.

- Returns

Nothing

setToDate(toDate)

Description

Set the end of the date range to return variances for if setVarianceScopeTypes ("Date_Range") is called.

Syntax

setToDate(toDate)

- Parameters

Date toDate - End date for the variance.

- Returns

Nothing

setVarianceEntryType(varianceType)

Description

Set the variance types to include in the results.

Info

Valid values are :

- RECIPE to return variances that occurred while a production item was selected to a recipe.
- SUB_RECIPE to return variances that occurred while a sub product code was selected for a production item. See Sub Recipes for more information.

Syntax

setVarianceEntryType(varianceType)

- Parameters

String varianceType - This include Recipe and Sub_Recipe as detailed in the Info section.

- Returns

Nothing

setVarianceScopeTypes(varianceScopeType)

Description

Set the variance scope to include in the results.

Info

Valid values are :

- Last - Return variances that occurred for the current or last recipe that a production item was set. This is useful for detecting any variances in real time for a production run. If the production run has stopped, it will return the variances as long as a new recipe has not been selected for the production item.
- Date_Range - Return variances for the date range specified with the setFromDate() and setToDate() properties.

Syntax

setVarianceScopeTypes(varianceScopeType)

- Parameters

String varianceScopeType - This include Last and Date_Range as detailed in the Info section.

- Returns

Nothing

Code Snippets

```
#Collection values we want to filter by
projectName = system.util.getProjectName()
itemPath = event.source.parent.getComponent('Production Line
Selector').selectedPathWithoutProject

#Build the filters object
filters = system.recipe.filter.variance.createNew()
filters.setProjectName(projectName)
filters.setVarianceEntryType("Recipe")
filters.setVarianceScopeTypes("Last")
filters.setItemPath(itemPath)
filters.setIncludeChildren(False)

#Request the variances for the given filters
ds = system.recipe.getRecipeVariances(filters)
event.source.parent.getComponent('Table').data = ds
```

9.6.7 Instrument Interface Objects

The Instrument Interface Module has a parsing engine that takes raw data received from an instrument and from it, extract the desired values. The extracted values can be used to set tags, populate SPC sample measurement values, populate tables, written to database tables and more. Because the extracted values come in various flavors and have various uses, the paring engine returns the extracted values in a ParseResults object.

This section defines the ParseResults object and how to access the extracted values.

Parse Results

A ParseResult object is available from the call to getParseResults() on the Serial Controller component.

Properties:

isValid()

Description

Returns true if all parse values exist and are valid.

Syntax

isValid()

- Parameters

None

- Returns

If true indicates that all parse values exist and are valid.

- Scope

All

isRequiredValid()

Description

Returns true if all required parse values exist and are valid.

Syntax

isRequiredValid()

- Parameters

None

- Returns

If true indicates that all required parse values exist and are valid.

- Scope

All

get(type)

Description

Returns a list ParseValue objects of type specified by the parseValueType parameter.

Syntax

get(type)

- Parameters

ParseValueType type - Type of the parse value. Details are given in the Info section shown below.

- Returns

A list ParseValue objects of type specified by the parseValueType parameter.

- Scope

All

Info

Available parseValueType options:

1. A single, discrete value.

system.instrument.parse.types.SingleValue

2. A collections of ParseRow objects.

system.instrument.parse.types.RowCollection

Methods:

getAll()

Description

Returns a list of all ParseValue objects.

Syntax

getAll()

- Parameters

None

- Returns

A list of all ParseValue objects.

- Scope

All

getValue()

Description

Returns a ParseValue object for the parsed value specified by the name parameter. The name must match one of the names assigned to a parsing box defined in the parsing template.

Syntax

getValue(name)

- Parameters

Name

- Returns

The ParseValue object for the parsed value specified by the name parameter.

- Scope

All

getRowCollection()

Description

Returns a ParseRowCollection object for the name specified by the name parameter. The name must match one of the names assigned to a parsing box defined in the parsing template.

Syntax

getRowCollection(name)

- Parameters

Name

- Returns

A ParseRowCollection object for the name specified by the name parameter.

- Scope

All

createDataset()

Description

Returns a Dataset object for the parsed value specified by the name parameter. The name must match one of the names assigned to a parsing box defined in the parsing template. This supports converting a ParseRowCollection that is a result of either a CSV Column Parsing Box or a CSV Row Parsing Box into a Dataset. Dataset can be used to display the data in Table or other components in Ignition.

Syntax

createDataset(name)

- Parameters

Name

- Returns

A Dataset object for the parsed value specified by the name parameter.

- Scope

All

createValueMap()

Description

Returns a Map object containing name value pairs for all parsed values. The Map can be sent to the SPC module's Sample Entry component to automate populating sample measurement values from an instrument. The Map can also be accessed using scripting.

Syntax

createValueMap(valueName)

- Parameters

String valueName

- Returns

A Map object containing name value pairs for all parsed values.

- Scope

All

Parse Row

A ParseRow object is available from the getParseRows() function of the ParseRowCollection object.

Properties:

getParseValues()

Description

Results all of the parse values contained in the row.

Syntax

getParseValues()

- Parameters

None

- Returns

List of ParseValue objects.

- Scope

All

isRequiredValid()

Description

Checks to see if all parse value objects are both required and valid.

Syntax

isRequiredValid()

- Parameters

None

- Returns

If true indicates that all parse values objects that are required are valid.

- Scope

All

isValid()

Description

Checks to see if all parse value objects are valid.

Syntax

isValid()

- Parameters

None

- Returns

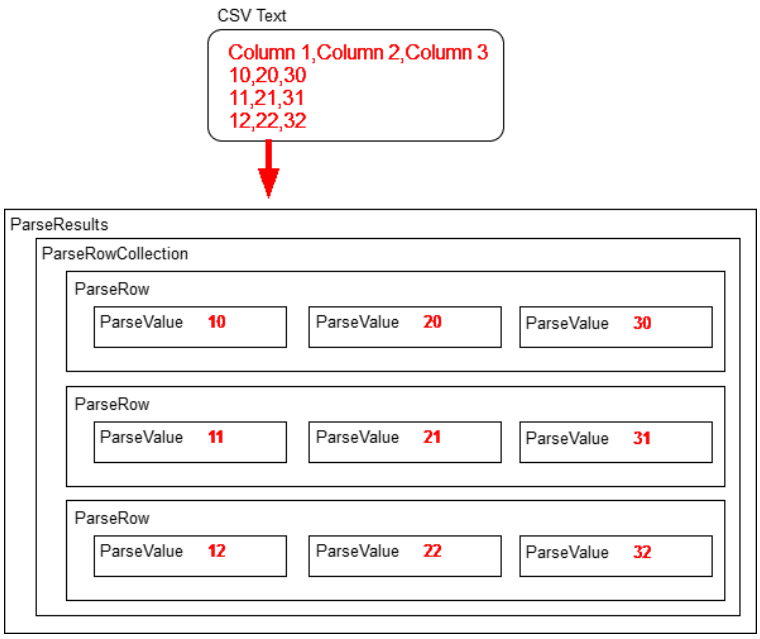
If true indicates that all parse value objects are valid.

- Scope

All

Parse Row Collection

The Parse Row Collection object contains one or more ParseRow objects. Each ParseRow object contains one or more ParseValue objects. When results contain values from a CSV source, there are rows and columns. As the image below depicts, CSV data is transformed into a ParseResults object.



Properties:

isValid()

Description

Returns true if all parse values within all parse rows are valid.

Syntax

isValid()

- Parameters

None

- Returns

If true indicates that all parse values within all parse rows are valid.

- Scope

All

isRequiredValid()

Description

Returns true if all parse values within all parse rows are required and are valid.

Syntax

isRequiredValid()

- Parameters

None

- Returns

If true indicates that all parse values within all parse rows are required and are valid.

- Scope

All

isRequired()

Description

Returns true if at least one parse values within all parse rows is required.

Syntax

isRequired()

- Parameters

None

- Returns

If true indicates that at least one parse values within all parse rows is required.

- Scope

All

getParseRows()

Description

Returns a list of all parse rows contained in this collection.

Syntax

getParseRows()

- Parameters

None

- Returns

A List of ParseRow objects contained in this collection.

- Scope

All

Code Examples

```
#Sample script to cycle though all parse value contained in parse
rows:
from org.apache.log4j import Logger
log = Logger.getLogger("ParseResult")
fileStr = system.file.readFileAsString("C:\\Temp\\Test.csv")
parseResults = system.instrument.parse.parseText("CSV Test Column"
, fileStr)
if parseResults.isValid():
 rowCollection = parseResults.getRowCollection("CSV Results")
 parseRowList = rowCollection.getParseRows()
 for parseRow in parseRowList:
         parseValueList = parseRow.getParseValues()
         for parseValue in parseValueList:
                 log.info("%s = %s" % (parseValue.getName(), str(p
arseValue.getValue())))
```

Parse SPC Results

Parse Value

A ParseValue object is available from the get method of the ParseResults object. Because parse values contain additional information such as units, data type, if it is required, etc, the value is contained in this object. Read the true value from the parse value uses getValue() function.

Properties:

getName()

Description

Gets the name of the parse value.

Syntax

getName()

- Parameters

None

- Returns

Name of this parse value.

- Scope

All

getType()

Description

Gets the type of this parse value.

Syntax

getType()

- Parameters

None

- Returns

The parse value type.

- Scope

All

getUnits()

Description

Gets the units extracted during parsing for this parse value. The Include Units option must be selected in the parse box options for the units to be extracted.

Syntax

getUnits()

- Parameters

None

- Returns

The units extracted during parsing for this parse value.

- Scope

All

getValue()

Description

Returns the true value of this parse value. For example, if the data type defined in the parse box options is a Float8, then a double will be returned.

Syntax

getValue()

- Parameters

None

- Returns

The exact value of this parse value.

- Scope

All

isRequired()

Description

Checks to see if a parse value is required.

Syntax

isRequired()

- Parameters

None

- Returns

If true indicates this parse value is required.

- Scope

All

isRequiredValid()

Description

Checks to see if a parse value is required and is valid.

Syntax

isRequiredValid()

- Parameters

None

- Returns

If true indicates this parse value is required and is valid.

- Scope

All

isValid()

Description

Checks to see if a parse value is valid.

Syntax

isValid()

- Parameters

None

- Returns

True, if parse value is valid.

- Scope

All

9.6.8 Web Service Objects

The Web Service Module has script functions and events that use various objects. This is because some web service information contains more data that can be represented with a single primitive data type. This section provide documentation of the methods and properties associated with these various objects.

After Run Event

This event is triggered by a web service listener after a call to run the web service.

Methods:

getOperation()

Description

Get the Web Service Operation that was run.

Syntax

getOperation()

- Parameters

None

- Returns

WSOperation operation - A Web Service Operation object containing the result of the Web Service.

- Scope

All

getResult()

Description

Gets the result of the web service operation.

Syntax

getResult()

- Parameters

None

- Returns

WSVariable result - The result obtained during the ws operation.

- Scope

All

Before Run Event

This event is triggered by a web service listener before a call to run the web service.

Methods:

getOperation()

Description

Gets the Web Service Operation.

Syntax

getOperation()

- Parameters

None

- Returns

WSOperation operation - The Web Service Operation object that represents the operation of the Web Service.

- Scope

All

Parse Error Event

This event is triggered by a web service listener if a call to run the web service generates a parsing error on the operation data.

Methods:

getErrorMessage()

Description

Gets error message from the Parse Error.

Syntax

getErrorMessage()

- Parameters

None

- Returns

String message - A String object describing the Parse Error.

- Scope

All

getException()

Description

Gets the exception for the parse error, if any.

Syntax

getException()

- Parameters

None

- Returns

WSException exception - The exception for the parse error occurred.

- Scope

All

getOperation()

Description

Gets the Web Service Operation that failed.

Syntax

getOperation()

- Parameters

None

- Returns

WSOperation operation - A Web Service Operation object representing the operation that failed.

- Scope

All

getVariablePath()

Description

Gets path to the Web Service Variable that failed to parse.

Syntax

getVariablePath()

- Parameters

None

- Returns

String path - A String object that represents the path to the Web Service Variable that failed to parse.

- Scope

All

setErrorMessage(message)

Description

Sets error message for the Parse Error.

Syntax

setErrorMessage(message)

- Parameters

String message - A String object describing the Parse Error.

- Returns

Nothing

- Scope

All

setException(exception)

Description

Sets the exception for the parse error.

Syntax

setException(exception)

- Parameters

WSException exception - The exception to set the parse error for.

- Returns

Nothing

- Scope

All

setVariablePath(path)

Description

Sets the path for Web Service Variable that failed to parse.

Syntax

setVariablePath(path)

- Parameters

String path - A String object that represents the path to the Web Service Variable that failed to parse.

- Returns

Nothing

- Scope

All

Service Error Event

This event is triggered by a web service listener if a call to run the web service results in a service error.

Methods:

getErrorMessage()

Description

The error message from the Web Service provider.

Syntax

getErrorMessage()

- Parameters

None

- Returns

String message - A String object describing the error from the Web Service provider.

- Scope

All

getOperation()

Description

The Web Service Operation that failed.

Syntax

getOperation()

- Parameters

None

- Returns

WSOperation - A Web Service Operation object representing the operation that failed.

- Scope

All

WS Element

The WSElement object is used to hold the individual elements defined in the schema for a web service that represent the header, body, input and output.

Properties:

addChild()

Description

This script function will add a child to the existing WSElement.

Syntax

addChild(child)

- Parameters

WSElement child - The child object to be added.

- Returns

Nothing

- Scope

All

getChild()

Description

Returns the WSElement at this index in the child WSElement list.

Syntax

getChild(index)

- Parameters

int index - The index of the child to be returned for.

- Returns

The child specified by the index.

- Scope

All

getChild(name)

Description

Retrieves a child specified by the name.

Syntax

getChild(childName)

- Parameters

String childName - The name of the child to be returned for.

- Returns

The WSElement associated with this child name.

- Scope

All

getChildNames()

Description

Returns all child names of the WSElement contained in this WSElement.

Syntax

getChildNames()

- Parameters

None

- Returns

List names - The list of names of all the children for this WSElement.

- Scope

All

getChildren()

Description

Returns a list of WSElements contained in this WSElement.

Syntax

getChildren()

- Parameters

None

- Returns

List children - All the children for this WSElement.

- Scope

All

getFullName()

Description

Returns a String representing the "." separated name of this WSElement and it's parent.

Syntax

getFullName()

- Parameters

None

- Returns

String name - The full name of this WSElement.

- Scope

All

getMaxOccurrences()

Description

Returns maximum occurrences this element can appear.

Syntax

getMaxOccurrences()

- Parameters

None

- Returns

int maxOcurrences - The integer representing the maximum number of occurrences.

- Scope

All

getName()

Description

Returns the name of this WSElement.

Syntax

getName()

- Parameters

None

- Returns

String name - The name of this WSElement.

- Scope

All

getParent()

Description

Returns the parent of this WSElement.

Syntax

getParent()

- Parameters

None

- Returns

WSElement parent - The parent for this WSElement.

- Scope

All

getQualifiedName()

Description

Returns the qualified name of this WSElement.

Syntax

getQualifiedName()

- Parameters

None

- Returns

String qualifiedName - The qualified name for this WSElement.

- Scope

All

getType()

Description

Returns WSType of this element.

Syntax

getType()

- Parameters

None

- Returns

The WSType for this element.

- Scope

All

getTypeName()

Description

Returns WSType of this element. Can be 'Complex', 'Simple' or 'Restricted'.

Syntax

getTypeName()

- Parameters

None

- Returns

String typeName - The type of this WSElement.

- Scope

All

isOptional()

Description

Returns true if this element is optional.

Syntax

isOptional()

- Parameters

None

- Returns

boolean optional - True if this element should be optional and False otherwise.

- Scope

All

setChildren(children)

Description

Sets a list of WSElements as the children for this WSElement.

Syntax

setChildren(children)

- Parameters

List children - Children to be set for this WSElement.

- Returns

Nothing

- Scope

All

setDataType(typeName)

Description

Sets the data type for the WSElement.

Syntax

setDataType(typeName)

- Parameters

String typeName - The data type to be set for. String is the default data type .

- Returns

Nothing

- Scope

All

setMaxOccurrences(maxOccurrences)

Description

Sets the maximum occurrences for this WSElement.

Syntax

setMaxOccurrences(maxOccurrences)

- Parameters

int maxOcurrences - The integer representing the maximum number of occurrences.

- Returns

Nothing

- Scope

All

setName()

Description

Sets the name of this WSElement.

Syntax

setName()

- Parameters

String name - The name of the WSElement to be set for.

- Returns

The name of the WSElement.

- Scope

All

setOptional()

Description

Set to true if this element should be optional or False otherwise.

Syntax

setOptional()

- Parameters

boolean optional - True if this element should be optional and False otherwise.

- Returns

Nothing

- Scope

All

setParent(parent)

Description

Sets the parent for this WSElement.

Syntax

setParent(parent)

- Parameters

WSElement parent - The parent for this WSElement.

- Returns

Nothing

- Scope

All

setQualifiedName(qualifiedName)

Description

Sets the qualified name for this WSElement.

Syntax

setQualifiedName(qualifiedName)

- Parameters

String name - The qualified name for this WSElement.

- Returns

Nothing

- Scope

All

setType(type)

Description

Returns WSType of this element.

Syntax

setType(type)

- Parameters

WSType type - The type to be set for.

- Returns

Nothing

- Scope

All

setTypeName(typeName)

Description

Returns WSType of this element. Can be 'Complex', 'Simple' or 'Restricted'.

Syntax

setTypeName(typeName)

- Parameters

String typeName - The type to be set for this WSElement.

- Returns

The WS type for this element.

- Scope

All

sortChildren()

Description

Sorts the WSElements to simple and complex children.

Syntax

sortChildren()

- Parameters

None

- Returns

Nothing

- Scope

All

WS Exception

The WSException object is returned from a web service call to handle errors and other exceptional events.

Methods:

getMessage()

Description

Returns a message with details about this web service exception.

Syntax

getMessage()

- Parameters

None

- Returns

String message - A message about the exception that occured.

- Scope

All

getVariable()

Description

Gets the web service variable associated with this exception.

Syntax

getVariable()

- Parameters

None

- Returns

WSVariable variable - The WSVariable object is returned from this web service.

- Scope

All

WS Operation

The WSOperation object holds all the operation data associated with a web service port and is available to the web service listener events via a call to event.getOperation(). It is available for more advanced usage of a web service.

Properties:

addBodyElement(bodyPart)

Description

Adds body part of the WS element to this WS operation. The WS element has a header part, a body part and the result part.

Syntax

addBodyElement(bodyPart)

- Parameters

WSElement bodyPart - The body part of the WS element to be added.

- Returns

Nothing

- Scope

All

addHeaderElement(headerPart)

Description

Adds header part of the WS element to this WS operation.

Syntax

addHeaderElement(headerPart)

- Parameters

WSElement headerPart - The header part of the WS element to be added.

- Returns

Nothing

- Scope

All

addResultElement(resultPart)

Description

Adds result part of the WS element to this WS operation.

Syntax

addResultElement(resultPart)

- Parameters

WSElement resultPart - The result part of the WS element to be added.

- Returns

Nothing

- Scope

All

copy()

Description

Creates a copy of this WS operation.

Syntax

copy()

- Parameters

None

- Returns

WSOperation clone - The copy of this WS operation.

- Scope

All

getBodyElement()

Description

Gets the body part of the WS element associated with this WS operation.

Syntax

getBodyElement()

- Parameters

None

- Returns

WSElement bodyElement - The body part of WS element corresponding to this web service operation.

- Scope

All

getBodyVariable()

Description

Returns the body variable of this WS operation.

Syntax

getBodyVariable()

- Parameters

None

- Returns

WSVariable bodyVariable - The body variable corresponding to this WS operation.

- Scope

All

getFullName()

Description

Returns the full name of this operation.

Syntax

getFullName()

- Parameters

None

- Returns

String name - The full name of this WS operation.

- Scope

All

getHeadersElement()

Description

Gets the headers element associated with this WS operation.

Syntax

getHeadersElement()

- Parameters

None

- Returns

WSElement headersElement - The headers element for this WS operation.

- Scope

All

getHeadersVariable()

Description

Returns the headers variable for this WS operation.

Syntax

getHeadersVariable()

- Parameters

None

- Returns

WSVariable headersVariable - The headers variable for this WS operation.

- Scope

All

getInput()

Description

Gets the input for this WS operation.

Syntax

getInput()

- Parameters

None

- Returns

S tring input - The input for this WS operation.

- Scope

All

getName()

Description

Gets the name of this WS operation.

Syntax

getName()

- Parameters

None

- Returns

String name - The name of this WS operation.

- Scope

All

getOutput()

Description

Returns the output of this WS operation.

Syntax

getOutput()

- Parameters

None

- Returns

String output - The output of this WS operation.

- Scope

All

getPort()

Description

Returns the port of this WS operation.

Syntax

getPort()

- Parameters

None

- Returns

WSPort port - The web service port to perform this operation.

- Scope

All

getResult()

Description

Gets the WS variable created by this WS operation.

Syntax

getResult()

- Parameters

None

- Returns

WSVariable results - The results of this WS operation.

- Scope

All

getResultsElement()

Description

Gets the result element of this operation.

Syntax

getResultsElement()

- Parameters

None

- Returns

WSElement resultsElement - The result element of this operation.

- Scope

All

getResultsVariable()

Description

Gets the results variable of this operation.

Syntax

getResultsVariable()

- Parameters

None

- Returns

WSVariable resultsVariable - The results variable of this operation.

- Scope

All

getWebFault()

Description

Gets the fault associated with this web service operation.

Syntax

getWebFault()

- Parameters

None

- Returns

String webFault - The message explaining the fault occurred during this web service operation.

- Scope

All

hasWebFault()

Description

Checks whether this WSOperation has any faults.

Syntax

hasWebFault()

- Parameters

None

- Returns

boolean - True, if there exists any faults and False otherwise.

- Scope

All

setBodyElement(bodyElement)

Description

Sets the body part of the WS element associated with this WS operation.

Syntax

setBodyElement(bodyElement)

- Parameters

WSElement bodyElement - The body part to set WS operation for.

- Returns

Nothing

- Scope

All

setBodyVariable(bodyVariable)

Description

Sets the body variable of this WS operation.

Syntax

setBodyVariable(bodyVariable)

- Parameters

WSVariable bodyVariable - The body variable to set WS operation for.

- Returns

Nothing

- Scope

All

setHeadersElement(headersElement)

Description

Sets the headers element associated with this WS operation.

Syntax

setHeadersElement(headersElement)

- Parameters

WSElement headersElement - The headers element to set WS operation for.

- Returns

Nothing

- Scope

All

setHeadersVariable(headersVariable)

Description

Sets the headers variable for this WS operation.

Syntax

setHeadersVariable(headersVariable)

- Parameters

WSVariable headersVariable - The headers variable for this WS operation.

- Returns

Nothing

- Scope

All

setInitialized(initialized)

Description

Return true if this WS operation have been initialized.

Syntax

setInitialized(initialized)

- Parameters

b oolean initialized - Set to T rue if this WS operation have been initialized, else False.

- Returns

Nothing

- Scope

All

setInput(input)

Description

Set input for this operation.

Syntax

setInput(input)

- Parameters

b oolean input - The input to set for.

- Returns

Nothing

- Scope

All

setModified(modified)

Description

Sets whether this operation has been modified.

Syntax

setModified(modified)

- Parameters

b oolean modified - True if this operation has been modified and False otherwise.

- Returns

Nothing

- Scope

All

setName(name)

Description

Sets the name of this WS operation.

Syntax

setName(name)

- Parameters

String name - The name of this WS operation.

- Returns

Nothing

- Scope

All

setOutput(output)

Description

Sets the output of this WS operation.

Syntax

setOutput(output)

- Parameters

String output - The output of this WS operation.

- Returns

Nothing

- Scope

All

setPort(port)

Description

Sets the port for this WS operation.

Syntax

setPort(port)

- Parameters

String port - The port for this WS operation.

- Returns

Nothing

- Scope

All

setResultsElement(resultsElement)

Description

Sets the result elements for this WS operation.

Syntax

setResultsElement(resultsElement)

- Parameters

String resultsElement - The results element to set the operation for.

- Returns

Nothing

- Scope

All

setResultsVariable(resultsVariable)

Description

Sets the WS variable object that contains the results of this operation.

Syntax

setResultsVariable(resultsVariable)

- Parameters

WSVariable resultsVariable - The results variable to set operation for.

- Returns

Nothing

- Scope

All

setWebFault(message)

Description

Sets a message when a fault occurs in this WS operation.

Syntax

setWebFault(message)

- Parameters

String message - The message to set for web faults.

- Returns

Nothing

- Scope

All

setWrapped(wrapped)

Description

Return true if this WS operation have been wrapped.

Syntax

setWrapped(wrapped)

- Parameters

boolean wrapped - Set to T rue if this WS operation have been wrapped, else False.

- Returns

Nothing

- Scope

All

Code Example

```
# Code snippet from the After Run event of a web service listener
wsOperation = event.getOperation()
if wsOperation.hasWebFault():
   log.error(wsOperation.getWebFault())
```

Overview

WSOptions class is for backwards compatibility

getAuthType()

Description

Gets the authentication type associated with this web service.

Syntax

getAuthType()

- Parameters

None

- Returns

String authType - The authentication type associated with this web service.

- Scope

All

getAuthTypeName()

Description

Gets the authentication type name associated with this web service.

Syntax

getAuthTypeName()

- Parameters

None

- Returns

String authType - The short authentication type name associated with this web service.

- Scope

All

getAuthTypesList()

Description

Gets the list of authentication types associated with this web service.

Syntax

getAuthTypesList()

- Parameters

None

- Returns

List<String> authType - The list of authentication types associated with this web service.

- Scope

All

getPassword()

Description

Returns the password set for the web service.

Syntax

getPassword()

- Parameters

None

- Returns

String password - The password set for this web service.

- Scope

All

getTimeout()

Description

Gets the timeout in seconds for the web service.

Syntax

getTimeout()

- Parameters

None

- Returns

int timeout - Optional timeout in seconds.

- Scope

All

getURL()

Description

Gets the URL associated with this web service.

Syntax

getURL()

- Parameters

None

- Returns

String url - The url for this web service.

- Scope

All

getUserName()

Description

Gets the user name associated with this web service.

Syntax

getUserName()

- Parameters

None

- Returns

String userName - The user name for this web service.

- Scope

All

setAuthType(authTypeDisplayName)

Description

Sets the authentication type for this web service.

Syntax

setAuthType(authTypeDisplayName)

- Parameters

String authTypeDisplayName - The short authentication type name to be for this web service.

- Returns

Nothing

- Scope

All

setPassword(password)

Description

Sets the password for the web service.

Syntax

setPassword(password)

- Parameters

String password - The password to set for.

- Returns

Nothing

- Scope

All

setTimeout(timeout)

Description

Sets the timeout associated with this web service.

Syntax

setTimeout(timeout)

- Parameters

int timeout - The timeout in seconds to be set.

- Returns

Nothing

- Scope

All

setURL(url)

Description

Sets the URL associated with this web service.

Syntax

setURL(url)

- Parameters

String url - The url for this web service.

- Returns

Nothing

- Scope

All

setUserName(userName)

Description

Sets the user name for the web service.

Syntax

setUserName(userName)

- Parameters

String userName - The user name to be set.

- Returns

Nothing

- Scope

All

WS Port

addOperation(operation)

Description

This script function will add an operation to the web service port.

Syntax

addOperation(operation)

- Parameters

WSOperation operation - The operation to be added.

- Returns

Nothing

- Scope

All

getDefinition()

Description

Returns the WS definition for this port.

Syntax

getDefinition()

- Parameters

None

- Returns

WSDefinition definition - The definition corresponding to this WS port.

- Scope

All

getName()

Description

Gets the name of this WS operation.

Syntax

getName()

- Parameters

None

- Returns

String name - The name of this operation.

- Scope

All

getOperation(name)

Description

Returns the operation specified by the name parameter.

Syntax

getOperation(name)

- Parameters

String name - The name of the operation to return for.

- Returns

WSOperation operation - The web service operation with the specified name.

- Scope

All

getOperationNames()

Description

Returns a list of operation names.

Syntax

getOperationNames()

- Parameters

None

- Returns

List<String> operationNames - A list of the names of operations associated with this WS port.

- Scope

All

getOperations()

Description

Gets all the operations associated with this WS port.

Syntax

getOperations()

- Parameters

None

- Returns

WSOperation operations - A list of operations for this web service port.

- Scope

All

setDefinition(definition)

Description

Sets the definition for this WS port.

Syntax

setDefinition(definition)

- Parameters

WSDefinition definition - The web service definition to set for.

- Returns

Nothing

- Scope

All

setName(name)

Description

Sets a name to this ws operation.

Syntax

setName(name)

- Parameters

String name - The name to set the ws operation for.

- Returns

Nothing

- Scope

All

setOperations(operations)

Description

Set the operations for this WS operation.

Syntax

setOperations(operations)

- Parameters

WSOperation operations - A list of operations for this web service port.

- Returns

Nothing

- Scope

All

WS Variable

The WSVariable object is returned from a web service call to hold the results of the call. This object contains references to child WSVariables that are defined in the schema of the output of the web service call.

Properties:

addChild(index, child)

Description

Adds a new child to this ws variable.

Syntax

addChild(index, child)

- Parameters

int index - The index for new child.

WSVariable child - The variable to add as a child to this WS variable.

- Returns

Nothing

- Scope

All

addChild(childElement)

Description

Adds and returns a new WSElement instance of childType to children.

Syntax

addChild(childElement)

- Parameters

WSElement childElement - The web service element to be added as a child.

- Returns

Nothing

- Scope

All

addChild(child)

Description

Adds a new child object to this WS variable.

Syntax

addChild(child)

- Parameters

WSVariable child - The WS variable to be added as child.

- Returns

Nothing

- Scope

All

clear()

Description

Removes this WS variable.

Syntax

clear()

- Parameters

None

- Returns

Nothing

- Scope

All

copy()

Description

Creates an exact copy of the ws variable and also a copy of its children.

Syntax

copy()

- Parameters

None

- Returns

Returns the new copy of the WS variable.

- Scope

All

getBindType()

Description

Gets the type of the bind of this WS variable.

Syntax

getBindType()

- Parameters

None

- Returns

BindType bindType - The bind type of this web service variable.

- Scope

All

getChild(childElement)

Description

Returns the child of this WS variable.

Syntax

getChild(childElement)

- Parameters

WSElement childElement - The WS element name to filter results.

- Returns

The child of this WSVariable.

- Scope

All

getChild(element, index)

Description

Returns the child of this WS variable.

Syntax

getChild(element, index)

- Parameters

WSElement element - The WS element name to filter results.

int index - The index of child to return for.

- Returns

The child of this WSVariable.

- Scope

All

getChild(elementName, index)

Description

Returns the child of this WS variable.

Syntax

getChild(elementName, index)

- Parameters

int index - The index of child to return for.

String elementName - The children type to return for.

- Returns

The child of this WSVariable.

- Scope

All

getChild(index)

Description

Returns the child of this WSVariable specified by the index parameter.

Syntax

getChild(index)

- Parameters

int index - The index of child to return for.

- Returns

The child of this WS variable with the given index.

- Scope

All

getChild(name)

Description

Returns the WSVariable associated with this child name.

Syntax

getChild(name)

- Parameters

String childName - The name of the child to be returned.

- Returns

WSVariable specified by the name parameter.

- Scope

All

getChildCount()

Description

Gets the number of child elements contained by this WS variable. If it doesn't have a child, a count of zero is returned.

Syntax

getChildCount()

- Parameters

None

- Returns

The number of children for this variable.

- Scope

All

getChildInstances(element)

Description

Gets the child instances of the type specified by the WSElement.

Syntax

getChildInstances(element)

- Parameters

WSElement element - The WS element object to filter the results.

- Returns

List<WSVariable> The child instances associated with this WS variable.

- Scope

All

getChildInstances(elementName)

Description

Gets the child instances of the type specified by the elementName parameter.

Syntax

getChildInstances(elementName)

- Parameters

String elementName - The WS element name to filter the results.

- Returns

List<WSVariable> The child instances associated with this WS variable.

- Scope

All

getChildNames()

Description

Returns all child element names contained in this WSVariable.

Syntax

getChildNames()

- Parameters

None

- Returns

ArrayList<String> - The names of the children of this WSVariable.

- Scope

All

getChildOccurrences(elementName)

Description

Returns the number of occurrences of the specified child.

Syntax

getChildOccurrences(elementName)

- Parameters

String elementName - The name of the element to return the count for.

- Returns

int count - The number of occurrences of the specified child.

- Scope

All

getChildOccurrences(childElement)

Description

Returns the number of occurrences of the specified child element.

Syntax

getChildOccurrences(childElement)

- Parameters

WSElement childElement - The child element to return the count for.

- Returns

int count - The number of occurrences of the specified child element.

- Scope

All

getChildren()

Description

Returns a list of children of this WS variable.

Syntax

getChildren()

- Parameters

None

- Returns

List<WSVariable> The list of children of this WS variable.

- Scope

All

getElement()

Description

Returns a WSElement Object representing the schema element of the child.

Syntax

getElement()

- Parameters

None

- Returns

WSElement - The WS element associated with this WS variable.

- Scope

All

getExpression()

Description

Returns the expression set for binding property.

Syntax

getExpression()

- Parameters

None

- Returns

Object expression - The expression set for this WS variable.

- Scope

All

getName()

Description

Gets the name of this WS variable.

Syntax

getName()

- Parameters

None

- Returns

String name - The name of WS variable.

- Scope

All

getParent()

Description

Gets the parent of this web service variable.

Syntax

getParent()

- Parameters

None

- Returns

WSVariable parent - The parent corresponding to this WS variable.

- Scope

All

getReasonPhrase()

Description

Get the reason phrase.

Syntax

getReasonPhrase()

- Parameters

None

- Returns

String reasonPhrase - The corresponding reason phrase.

- Scope

All

getResponseTime()

Description

Get the response time in milliseconds.

Syntax

getResponseTime()

- Parameters

None

- Returns

Long responseTime - The response time in milliseconds.

- Scope

All

getStatusCode()

Description

Get the associated status code.

Syntax

getStatusCode()

- Parameters

None

- Returns

int statusCode - The corresponding status code.

- Scope

All

getValue()

Description

Returns an Object representing the value of the child.

Syntax

getValue()

- Parameters

None

- Returns

Object value - The value of the child of this WSVariable.

- Scope

All

hasChild(childElement)

Description

Boolean indicating the presence of child for the given web service element.

Syntax

hasChild(childElement)

- Parameters

WSElement childElement - The WS element to check the existence of child for.

- Returns

boolean - True, if there exist a child for the childElement and False otherwise.

- Scope

All

hasChildren()

Description

Boolean indicating that the web service element has children.

Syntax

hasChildren()

- Parameters

WSElement childElement - The WS element to check the existence of child for.

- Returns

boolean - True, if the WS element has children and False otherwise.

- Scope

All

isArray()

Description

This method returns true if this object represents an array class, else false.

Syntax

isArray()

- Parameters

None

- Returns

boolean - True, if the WS variable class is an array and False otherwise.

- Scope

All

isBound()

Description

Boolean indicating that this WS variable is bound.

Syntax

isBound()

- Parameters

None

- Returns

boolean True, if the WS variable is bound and False otherwise.

- Scope

All

isValid()

Description

Checks whether this WS variable is valid or not.

Syntax

isValid()

- Parameters

None

- Returns

boolean True if the WS variable is valid and False otherwise.

- Scope

All

recordOrginalValues()

Description

Records the original values of the WS variable.

Syntax

recordOriginalValues()

- Parameters

None

- Returns

Nothing

- Scope

All

removeChild(childElement)

Description

Removes the child from this WS variable.

Syntax

removeChild(childElement)

- Parameters

WSElement childElement - The WS element to be removed.

- Returns

Nothing

- Scope

All

removeChild(childVariable)

Description

Removes the child from the WS Variable.

Syntax

removeChild(childVariable)

- Parameters

WSVariable childVariable - The WS child variable to be removed.

- Returns

Nothing

- Scope

All

removeChild(index)

Description

Removes the child from the WS variable.

Syntax

removeChild(index)

- Parameters

int index - The index of child to be removed.

- Returns

Nothing

- Scope

All

reset()

Description

Clears all the children and resets the value to the original value.

Syntax

reset()

- Parameters

None

- Returns

Nothing

- Scope

All

setBindType(bindType)

Description

Sets the type of bind associated with this WS variable.

Syntax

setBindType(bindType)

- Parameters

BindType bindType - The bind type to bound this WS variable.

- Returns

Nothing

- Scope

All

setChildren(children)

Description

Sets the children for this WS variable.

Syntax

setChildren(children)

- Parameters

List<WSVariable> children - The list of WS variable to be set as children.

- Returns

Nothing

- Scope

All

setElement(element)

Description

Sets a WS element for this WS variable.

Syntax

setElement(element)

- Parameters

WSElement element - The element set the WS variable for.

- Returns

Nothing

- Scope

All

setExpression(expression)

Description

Sets the expression for binding.

Syntax

setExpression(expression)

- Parameters

Object expression - The expression to set the WS variable for.

- Returns

Nothing

- Scope

All

setName(name)

Description

Sets the name for the WS variable.

Syntax

setName(name)

- Parameters

String name - The name to set the WS variable for.

- Returns

Nothing

- Scope

All

setParent(parent)

Description

Sets the parent for the WS variable.

Syntax

setParent(parent)

- Parameters

WSVariable parent - The object to set the WS variable for.

- Returns

Nothing

- Scope

All

setValue(value)

Description

Sets the value for the WS variable.

Syntax

setValue(value)

- Parameters

Object value - The value to set for the WS variable.

- Returns

Nothing

- Scope

All

Code Example

```
wsVariable = system.ws.runWebService('Temperature Convert')
    resultData = wsVariable.getChild('FahrenheitToCelsiusResult')
    print  'value=%s' %(resultData.getValue() )
```

Methods:

toDataset()

Description

Converts a web service variable into a dataset.

Syntax

toDataset()

- Parameters

None

- Returns

Dataset - A dataset representing the given Web Service Variable.

- Scope

All

toDataset(expanded)

Description

Converts a web service variable into a dataset.

Syntax

toDataset(expanded)

- Parameters

boolean expanded - If set to 1 (true) then every child will be converted, if set to 0 then only the top level child will be converted.

- Returns

Dataset - A dataset representing the given Web Service Variable.

- Scope

All

toDict()

Description

Converts a web service variable into a python dictionary.

Syntax

toDict()

- Parameters

None

- Returns

PyDictionary pyDict - A python dictionary representing the given Web Service Variable.

All

toExpandedDataset()

Description

Returns a Dataset Object representing the child values of the WSVariable and iterates through any children of the child values.

Syntax

toExpandedDataset()

- Parameters

None

- Returns

Dataset - The dataset containing the child values of this WS variable.

- Scope

All

9.6.9 Barcode Objects

The Barcode Scanner Module has script functions and events that use various objects. The following sections provide documentation of the methods and properties associated with these various objects.

Barcode Event

The BarcodeEvent object is an object that contains the decoded results of a barcode. It is passed in the onBarcodeReceived event of the BarcodeScanner component and the system.barcode.scanner.decode script function.

Properties:

getErrorMessage()

Description

Returns the error message from the decoding process.

Syntax

getErrorMessage()

- Parameters

None

- Returns

Error message from the decoding process.

- Scope

All

getRawBarcode()

Description

Get the raw barcode as a string.

Syntax

getRawBarcode()

- Parameters

None

- Returns

The raw barcode.

- Scope

All

getResults()

Description

Returns a java hash table of the decoded results. The hash table has entries that have a key and a value. The key is defined in the configuration patterns and the value is itself a list of values returned from the Regex search.

For example, the pre configured GS1 pattern GTIN that has a key of “GS1-01” and a Regex pattern of “(01)(\d{14})”. When this pattern is found in the raw barcode, then an entry will be put into the match results with a key of “GS1-01” and the value will be a list of strings with 2 elements. The list of value strings is variable and is defined by the Regex pattern's grouping. In this case the first [0] element is “01” and the second [1] element contains the 14 digit GTIN number.

Syntax

getResults()

- Parameters

None

- Returns

A hash table of the decoded results.

- Scope

All

getUnmatched()

Description

Get the string of unmatched raw barcode after the decode method was called.

Syntax

getUnmatched()

- Parameters

None

- Returns

The unmatched raw barcode after the decode method was called.

- Scope

All

hasErrorMessage()

Description

Returns true if there is an error message passed with this event.

Syntax

hasErrorMessage()

- Parameters

None

- Returns

True, if there is an error message passed with this event.

- Scope

All

hasResults()

Description

Returns true if there is decoded barcode results that matched the patterns given.

Syntax

hasResults()

- Parameters

None

- Returns

T rue, if there is decoded barcode results that matched the patterns given.

- Scope

All

hasUnmatched()

Description

Returns true if there is an unmatched portion of the raw barcode after the decode method was called.

Syntax

hasUnmatched()

- Parameters

None

- Returns

True, if there is an unmatched portion of the raw barcode after the decode method was called.

- Scope

All

toDict()

Description

Returns the same results that getResults() but converted to python dictionary object with a string key and the value as an array of strings.

Syntax

toDict()

- Parameters

None

- Returns

A Python dictionary object containing the decoded results.

- Scope

All

Methods:

gs1ConvertToDate(value)

Description

This is helper function to convert a GS1 formatted barcode date to a date object.

Syntax

gs1ConvertToDate(value)

- Parameters

String value - String value in the format of “YYMMDD”

- Returns

The resultant date object.

- Scope

All

gs1ConvertToDouble(value, decimalPlace)

Description

This is a helper function to convert a GS1numeric value to a double with the correct decimal place.

Syntax

gs1ConvertToDouble(value, decimalPlace)

- Parameters

String value - String that represents the number to be converted to a double .

String decimalPlace - String that represents the place in the value parameter to place the decimal place.

- Returns

The resultant double value.

- Scope

All

gs1ConvertToFloat(value, decimalPlace)

Description

This is a helper function to convert a GS1numeric value to a float with the correct decimal place.

Syntax

gs1ConvertToFloat(value, decimalPlace)

- Parameters

String value - String that represents the number to be converted to a float.

String decimalPlace - String that represents the place in the value parameter to place the decimal place.

- Returns

The resultant float value.

- Scope

All

Properties:

getKey()

Description

Gets the key corresponding to this barcode pattern.

Syntax

getKey()

- Parameters

None

- Returns

String key - The key associated with this barcode pattern.

- Scope

All

getName()

Description

Returns the name corresponding to this barcode pattern.

Syntax

getName()

- Parameters

None

- Returns

String name - The name associated with this barcode pattern.

- Scope

All

getRegexPattern()

Description

Gets the regex pattern for this barcode.

Syntax

getRegexPattern()

- Parameters

None

- Returns

String regexPattern - The regex pattern for this particular barcode pattern.

- Scope

All

setKey(key)

Description

Sets the key for this barcode pattern.

Syntax

setKey(key)

- Parameters

String key - The key to set this barcode pattern for.

- Returns

Nothing

- Scope

All

setName(name)

Description

Sets the name for this barcode pattern.

Syntax

setName(name)

- Parameters

String name - The name to set the barcode pattern for.

- Returns

Nothing

- Scope

All

setRegexPattern(pattern)

Description

Sets the regex pattern for this barcode pattern.

Syntax

setRegexPattern(pattern)

- Parameters

String pattern - The regex pattern to set the barcode for.

- Returns

Nothing

- Scope

All

Properties:

getErrorMessage()

Description

Returns the error message from the decoding process.

Syntax

getErrorMessage()

- Parameters

None

- Returns

String errorMessage - The message to display when an error occurs.

- Scope

All

getRawBarcode()

Description

Gets the raw barcode for this decode results.

Syntax

getRawBarcode()

- Parameters

None

- Returns

String rawBarcode - The raw barcode for this decode results.

- Scope

All

getResults()

Description

Returns the results as a hash table after the decoding process.

Syntax

getResults()

- Parameters

None

- Returns

Hashtable<String, List<String>> results - A hash table of the decoded results.

- Scope

All

getUnmatched()

Description

Returns the unmatched barcode for this decode results.

Syntax

getUnmatched()

- Parameters

None

- Returns

String unMatched - The unmatched raw barcode after the decode method was called.

- Scope

All

GS1ConvertToDate(value)

Description

This is helper function to convert a GS1 formatted barcode date to a date object.

Syntax

GS1ConvertToDate(value)

- Parameters

String value - String value in the format of “YYMMDD”

- Returns

The resultant date object.

- Scope

All

GS1ConvertToDouble(value, decimalPlace)

Description

This is a helper function to convert a GS1numeric value to a double with the correct decimal place.

Syntax

GS1ConvertToDouble(value, decimalPlace)

- Parameters

String value - String that represents the number to be converted to a double .

String decimalPlace - String that represents the place in the value parameter to place the decimal place.

- Returns

The resultant double value.

- Scope

All

GS1ConvertToFloat(value, decimalPlace)

Description

This is a helper function to convert a GS1numeric value to a float with the correct decimal place.

Syntax

GS1ConvertToFloat(value, decimalPlace)

- Parameters

String value - String that represents the number to be converted to a float.

String decimalPlace - String that represents the place in the value parameter to place the decimal place.

- Returns

The resultant float value.

- Scope

All

hasErrorMessage()

Description

Boolean indicating the presence of an error.

Syntax

hasErrorMessage()

- Parameters

None

- Returns

boolean True, if there is an error message passed with the decode results.

- Scope

All

hasResults()

Description

Checks whether there is decoded barcode results that matched the patterns given.

Syntax

hasResults()

- Parameters

None

- Returns

boolean T rue, if there is decoded barcode results that matched the patterns given.

- Scope

All

hasUnmatched()

Description

Boolean indicating the presence of unmatched barcodes.

Syntax

hasUnmatched()

- Parameters

None

- Returns

boolean True, if there is any unmatched barcode.

- Scope

All

9.7 Scripting Functions

The Sepasoft MES scripting API, which is available under the module name "system", is full of functions that are useful when designing Sepasoft MES projects. From starting production runs, analyzing production results, to importing or exporting production data, scripting functions can help. Some of these functions only work in the Gateway scope, and other only work in the Client scope, while the rest will work in any scope.

Also, there are script functions that are directly available on the MES objects. They support common tasks from setting material, personnel and other resources for production tasks to setting parenting relationships between material classes (categories) and material definitions.

For an overview and syntax of the scripting functions, see Scripting Overview and Syntax in the Ignition Documentation.

9.7.1

9.7.2 system.mes

There are many different types of MES objects in the Sepasoft MES system. All of these are inherited from the AbstractMESObject. Many of the scripting functions and properties refer to the common AbstractMESObject objects. The specific MES object types can be obtained by using the getMESObjectType() method on the object.

Code Snippet

```
filter = system.mes.object.filter.createFilter()
filter.setMESObjectNamePattern('Vinegar')
list = system.mes.searchMESObjects(filter)
for ndx in range(list.size()):
    mesObject = list.get(ndx)
    print mesObject.getMESObjectType().getDisplayName()
```

system.mes.abortOperation

Description

Abruptly end the current operation for the equipment specified by the equipmentPath parameter.

Syntax

system.mes.abortOperation(equipmentPath)

- Parameters

String equipmentPath - Equipment path of the equipment to end the operation.

- Returns

Nothing

- Scope

All

Warning, trace information may not be correctly recorded when this script function is used.

Code Examples

Code Snippet

```
#The following line of code will abort the current operation
response executing at Bottling Line 1. Different version.
system.mes.abortOperation('[global]\Dressings
Inc\California\Bottling\Bottling Line 1')
```

system.mes.abortSegment

Description

Abruptly end the specified response segment.

Syntax

system.mes.abortSegment(responseSegment)

- Parameters

MESResponseSegment responseSegment - The MES object to abort.

- Returns

Nothing

- Scope

All

Warning, trace information may not be correctly recorded when this script function is used.

Code Examples

Code Snippet

```
defSeg = system.mes.loadMESObject('Receive Material', 'Operatio
nsSegment')
eqPath = 'My Enterprise\California\Receiving\Unload Station 1'
respSeg = system.mes.createSegment(defSeg, eqPath)
system.mes.abortSegment(respSeg)
```

Code Snippet

```
#USE THIS METHOD IF THE SEGMENT'S (END OPERATION WHEN
COMPLETE) SETTING IS TRUE.
seg = system.mes.getActiveSegment('Dressings
Inc\California\Raw Materials\Unload Station 1', 'Unload
Balsamic Vinegar')
seg.abort()
system.mes.abortOperation('Dressings Inc\California\Raw
Materials\Unload Station 1')
```

Code Snippet

```
#USE THIS METHOD IF THE SEGMENT'S (END OPERATION WHEN
COMPLETE) SETTING IS FALSE.
oper = system.mes.getCurrentOperation('Dressings
Inc\California\Raw Materials\Unload Station 1')
seg = oper.getActiveSegment('Unload Balsamic Vinegar')
seg.end()
oper.end()
```

system.mes.addTagCollectorValue

Description

Record a single value for the MES tag collector. If a value has already been recorded for the same timestamp, an exception will be returned.

Syntax

system.mes.addTagCollectorValue(equipmentPath, collectorType, key, dateTime, value)

- Parameters

String equipmentPath - The path from the production model to the desired equipment.

String collectorType - The name of the tag collector type. See Tag Collector Types for more details.

String key - Where there are multiple i nstances of a tag collector type, this specifies which one to use . For example, there can be multiple MES counters for the "Equipment Count" tag collector type. If not needed, pass an empty string. In other words it is the name of the MES counter or the name of the Additional Factor.

Date dateTime - The date to the value being added.

Datatype value - The value to record to the MES tag collector.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#The following script will add tag value 2
dateTime = system.date.now()
equipmentPath = '[global]\Enterprise\San Marcos\MP Rotator\MP
Rotator 1'
collectorType = 'Equipment Mode'
key = ''
system.mes.addTagCollectorValue(equipmentPath, collectorType,
key, dateTime, 2)
```

system.mes.addTagCollectorValues

Description

Record multiple values for the MES tag collector. If a value has already been recorded for one of the timestamps, an exception will be returned.

Syntax

system.mes.addTagCollectorValues(equipmentPath, collectorType, key, values)

- Parameters

String equipmentPath - The path from the production model to the desired equipment.

String collectorType - The name of the tag collector type. See Tag Collector Types for more details.

String key - Where there are multiple i nstances of a tag collector type, this specifies which one to use . For example, there can be multiple MES counters for the "Equipment Count" tag collector type. If not needed, pass an empty string. In other words it is the name of the MES counter or the name of the Additional Factor.

PyDictionary values - A Python dictionary containing the date(of type Date) and value (Refer Datatype) pairs to add.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Define the from and to dates
date1 = system.date.getDate(2017, 2, 17)
dateTime1 = system.date.setTime(date1, 12, 55, 31)
date2 = system.date.getDate(2017, 2, 17)
dateTime2 = system.date.setTime(date2, 13, 06, 13)

equipmentPath = '[global]\Enterprise\San Marcos\MP Rotator\MP
Rotator 1'
collectorType = 'Equipment State'
key = ''
#Date and Value pairs as a python dictionary
values={dateTime1:4, dateTime2:3}

system.mes.addTagCollectorValues(equipmentPath, collectorType,
key, values)
```

system.mes.beginOperation

Descripton

Begin an operation at the equipment specified by the equipmentPath parameter. Once this function has been called, segments can begin for the equipment.

Syntax

system.mes.beginOperation(equipmentPath, operationsResponse)

- Parameters

String equipmentPath - Equipment path of where to run the operation.

MESOperationsReponse operationsResponse - The MES object to begin. Required properties must be set in the operations response object prior to calling this function.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
eqPath = 'My Enterprise\California\Receiving\Unload Station 1'
#load the operations definition
operDef = system.mes.loadMESObject('Receive Material', 'Operati
onsDefinition')
#create an operation for the operations definition
operResp = system.mes.createOperation(operDef)
#Begin the operation
system.mes.beginOperation(eqPath, operResp)
```

system.mes.beginSegment

Description

Begin the specified response segment.

Syntax

system.mes.beginSegment(responseSegment)

- Parameters

MESResponseSegment responseSegment - The MES object to begin. All required property values must be set prior to beginning.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Get the current operation
oper = system.mes.getCurrentOperation('Dressings
Inc\California\Raw Materials\Unload Station 1')

#Create a segment for the current operation
seg = oper.createSegment('Unload Balsamic Vinegar')
#Set material
seg.setMaterial('Vinegar', 'Balsamic Vinegar', 'Dressings
Inc\California\Raw Materials\Tank Farm\Vinegar Tank 1', 'TBV
1127', 100.0)
#Set personnel
seg.setPersonnel('Operator', 'Hechtman, Tom')
#Begin the segment
seg.begin()
```

Overview

These script functions are used to alter the category of operation schedules. One method for batch changes and one method for individual schedules.

Method Options

system.mes.changeScheduleCategory(fromDate, toDate, fromCategory, toCategory)

Description

Change the category of all operations schedule objects within the given date range with matching category.

Syntax

system.mes.changeScheduleCategory(fromDate, toDate, fromCategory, toCategory)

- Parameters

Date fromDate - The date to begin changing schedules.

Date toDate - The date to stop changing schedules.

String fromCategory - The schedule category to match before changing to the new category.

String toCategory - New category to assigned to the operation schedule.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Define the start and end dates
from java.util import Calendar
beginCal = Calendar.getInstance()
beginCal.add(Calendar.DAY_OF_MONTH, -30)
begin = beginCal.getTime()
endCal = Calendar.getInstance()
endCal.add(Calendar.DAY_OF_MONTH, -1)
end = endCal.getTime()
#Changes 'Held' to Active'
system.mes.changeScheduleCategory(begin, end, 'Held', 'Active')
```

system.mes.changeScheduleCategory(operationsScheduleUUID, toCategory)

Description

Change the category of an operations schedule object.

Syntax

system.mes.changeScheduleCategory(operationsScheduleUUID, toCategory)

- Parameters

Integer operationsScheduleUUID - The UUID of operationSchedule to change the category for.

String toCategory - New category to assigned to the operation schedule.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#This example would show how to change the ScheduleCategory to
'Held'
if(event.getMenuItemName() == 'Hold'):
    uuid = event.getScheduleEntry().
getMESOperationsScheduleLink().getMESObjectUUID()
    system.mes.changeScheduleCategory(uuid, 'Held')
```

Code Snippet

```
# This example would show how to change the ScheduleCategory
to 'Active'
elif(event.getMenuItemName() == 'Release Hold'):
uuid = event.getScheduleEntry().getMESOperationsScheduleLink().
getMESObjectUUID()
    system.mes.changeScheduleCategory(uuid, 'Active')
```

system.mes.copySchedule

Description

Copy an operations schedule and all associated operations requests and request segments objects.

Syntax

system.mes.copySchedule(operationsScheduleUUID)

- Parameters

String operationsScheduleUUID - The UUID of the operations schedule to copy.

- Returns

A list containing the operations schedule and all associated operations requests and request segments. The list is returned as a MES Object List object that is a collection holding MES objects.

- Scope

All

Code Snippet

```
#This example would show how to copy a schedule
system.mes.copySchedule('ca6ecd0b-d78d-40d4-9e89-41b5cf8e3f6b')
```

system.mes.createMESObject

Description

Returns a new instance of an MES object based on the mesObjectType parameter.

Syntax

system.mes.createMESObject(mesObjectTypeName)

- Parameters

String MES Object Type Name - The MES object type to base the new instance. This can be one of MES object types defined in MESObjectTypes. See MES Object Type Name for more details.

- Returns

A new instance of the MES object type specified in the mesObjectTypeName parameter.

- Scope

All

Code Examples

Code Snippet

```
#In the example below, a new MaterialClass MES object is
created.
#After it is created, the name is set then a custom property
is added to it and then it is saved.
matClass = system.mes.createMESObject('MaterialClass')
matClass.setPropertyValue('Name', 'Turkey')
matClass.addCustomProperty('Weight', 'Float8', 'Weight of the
turkey', 'Lbs', True, True)
system.mes.saveMESObject(matClass)
```

system.mes.createOperation

Description

Create a new MESOperationsResponse from the specified operations definition. This is done to create a new instance of a MESOperationsSegment object needed to begin an operation.

Syntax

system.mes.createOperation(operationsDefinitionName, equipmentPath)

- Parameters

String operationsDefinitionName - The name of the operations definition to base the operations response on.

String equipmentPath - Equipment path of where it will be run.

- Returns

A new MESOperationsResponse object.

- Scope

All

Code Examples

Code Snippet

```
#This code will create a new operation for the specified
equipment
eqPath = '[global]\My Enterprise\California\Receiving\Unload
Station 2'
system.mes.createOperation('Receive Turkeys', eqPath)
```

system.mes.createOperationFromRequestUUID

Description

Create a new MESOperationsResponse from the specified operations request UUID. When wanting to begin operations requests that have previously been scheduled, this method will create a new instance of a operation response from it.

Syntax

system.mes.createOperationFromRequestUUID(operationsRequestUUID)

- Parameters

String operationsRequestUUID - The UUID of the MES operations request object, which is its unique ID.

- Returns

A MESObjectList object holding all the objects associated with the new operations response.

- Scope

All

Code Examples

Code Snippet

```
#Prints the list of operation response objects
operResp = system.mes.createOperationFromRequestUUID('ccedca08-
7d74-4566-a47f-d1a8fe03e337')
for objs in operResp:
    print objs
```

Output

```
OperationsPerformance (09f79c29-a382-4a6b-871d-5f046586bef5,
Package Nuts Schedule, 0 parents, 0 children, 0 custom
properties, 1 complex properties)
OperationsResponse (d703c3f7-061e-4348-9fe3-f2db1c3050d9,
Package Nuts, 0 parents, 0 children, 0 custom properties, 1
complex properties)
```

system.mes.createSchedule

Description

Create a new MESOperationsSchedule from the specified operations definition that is used for scheduling. The Operations Schedule object will contain one or more linked Operations Requests, each with their associated Requests Segments.

Note that the objects must be saved to be made manifest in the system.

Syntax

system.mes.createSchedule(operationsDefinition)

- Parameters

MESOperationsDefinition operationsDefinition - The operations definition to base the root operations request on.

- Returns

A list containing a new operations schedule and all associated operations requests and request segments. The list is returned as a MESObjectList object that is a collection holding MES objects.

- Scope

All

Code Examples

Code Snippet

```
#Load the operation definition object
operationsDefinition = system.mes.loadMESObject('Receive
Turkeys', 'OperationsDefinition')

#Create a schedule for the specified operation
schedule = system.mes.createSchedule(operationsDefinition)
##Print each item for verification purposes
for ndx in range(schedule.size()):
    print schedule.get(ndx)

##Save the schedule objects to make them manifest in the
system.
system.mes.saveSchedule(schedule)
```

```
Output

OperationsSchedule (b15be936-9c94-4334-a1be-3ad97d115c2a,
Receive Turkeys Schedule, 0 parents, 0 children, 0 custom
properties, 1 complex properties)
OperationsRequest (ea82b3ea-6fe3-4a65-94a5-55e34c0ff231,
Receive Turkeys, 0 parents, 0 children, 0 custom properties, 2
complex properties)
RequestSegment (89f1698c-47f3-402e-a2d0-da49812d72f6, Receive
Turkeys, 0 parents, 0 children, 0 custom properties, 6 complex
properties)
```

Overview

Description

Create a new MESResponseSegment from the specified operations segment. This is done to create a new instance of a MESResponseSegement object needed to begin a segment.

These script functions are used to create a new response segment object.

Method Options

system.mes.createSegment(definitionSegment, equipmentPath)

Syntax

system.mes.createSegment(definitionSegment, equipmentPath)

- Parameters

AbstractMESObject definitionSegment - The operations segment to base the response segment on. See AbstractMESObject object in the MES documentation.

String equipmentPath - Equipment path of the equipment to use when creating the segment. This is required and is used when a material class is specified in the segment, it has to be replaced with the specific equipment.

- Returns

A new MESResponseSegment object.

- Scope

All

Code Examples

```
Code Snippet

#Creates a segment
obj = system.mes.loadMESObject('Receive Turkeys', 'OperationsSe
gment')
system.mes.createSegment(obj, 'My
Enterprise\California\Receiving\Unload Station 1')
```

system.mes.createSegment(definitionSegment, equipmentPath, autoAssignOptions)

Syntax

system.mes.createSegment(definitionSegment, equipmentPath, autoAssignOptions)

- Parameters

AbstractMESObject definitionSegment - The operations segment to base the response segment on. See AbstractMESObject object in the MES documentation.

String equipmentPath - Equipment path of the equipment to use when creating the segment. This is required and is used when a material class is specified in the segment, it has to be replaced with the specific equipment.

Boolean autoAssignOptions - If true, automatically assign material, lot and person options. Otherwise, they have to be set prior to beginning the segment.

- Returns

A new MESResponseSegment object.

- Scope

All

Code Examples

Code Snippet

```
#This code will create a new instance of the
MESResponseSegment object
seg = system.mes.createSegment('Load Assembly Tray', '[global]
\Dressings Inc\California\Assembly\PS Assembly', False)
seg.setMaterial('Housing', 'Housing', 'Assembly Tray 8')
seg.begin()
```

system.mes.createSegment(operationsResponse, segmentName, equipmentPath, autoAssignOptions)

Syntax

system.mes.createSegment(operationsResponse, segmentName, equipmentPath, autoAssignOptions)

- Parameters

MESOperationsResponse object - The operations response to create the response segment for.

String segmentName - The name of the operationsSegment to base the response segment on. If this property is empty, then the name of the operations response object will be used.

String equipmentPath - Equipment path of the equipment to use when creating the segment. This is required and is used when a material class is specified in the segment, it has to be replaced with the specific equipment.

Boolean autoAssignOptions - If true, automatically assign material, lot and person options. Otherwise, they have to be set prior to beginning the segment.

- Returns

A new MESResponseSegment object.

- Scope

All

Code Examples

Code Snippet

```
#Creates a segment for a specific operation
eqPath = 'My Enterprise\California\Receiving\Unload Station 1'
operDef = system.mes.loadMESObject('Receive Turkeys', 'Operatio
nsDefinition')
operResp = system.mes.createOperation(operDef)
system.mes.createSegment(operResp, 'Receive Turkeys', eqPath, T
rue)
```

system.mes.createSegment(operationResponse, segmentName, autoAssignOptions)

Syntax

system.mes.createSegment(operationsResponse, segmentName, autoAssignOptions)

- Parameters

MESOperationsResponse operationsResponse - The operations response to create the response segment for.

String segmentName - The name of the operationsSegment to base the response segment on. If this property is empty, then the name of the operations response object will be used.

Boolean autoAssignOptions - If true, automatically assign material, lot and person options. Otherwise, they have to be set prior to beginning the segment.

- Returns

A new MESResponseSegment object.

- Scope

All

Code Examples

Code Snippet

```
#Load an operation definition object
operDef = system.mes.loadMESObject('Receive Turkeys', 'Operatio
nsDefinition')
#Create an operation for the operations definition object
operResp = system.mes.createOperation(operDef)
#Create a segment for the specified operation
system.mes.createSegment(operResp, 'Receive Turkeys', True)
```

system.mes.createSegment(operationsSegmentName, equipmentPath, autoAssignOptions)

Syntax

system.mes.createSegment(operationsSegmentName, equipmentPath, autoAssignOptions)

- Parameters

String operationsSegmentName - The name of the operations segment to base the response segment on.

String equipmentPath - Equipment path of the equipment to use when creating the segment. This is required and is used when a material class is specified in the segment, it has to be replaced with the specific equipment.

Boolean autoAssignOptions - If true, automatically assign material, lot and person options. Otherwise, they have to be set prior to beginning the segment.

- Returns

A new MESResponseSegment object.

- Scope

All

Code Examples

Code Snippet

```
#Creates a segment for a given equipment path and segment name
system.mes.createSegment('Receive Turkeys', 'My
Enterprise\California\Receiving\Unload Station 1', True)
```

system.mes.createSegmentForOperation

Description

Create a new MESResponseSegment from the specified operations segment. This is done to create a new instance of a MESResponseSegement object needed to begin a segment.

Syntax

system.mes. createSegmentForOperation (operationsResponseUUID, segmentName, autoAssignOptions)

- Parameters

String operationsResponseUUID - The operations response UUID to create the response segment for.

String segmentName - The name of the operations segment to base the response segment on. If this property is empty, then the name of the operations response object will be used.

Boolean autoAssignOptions - If true, automatically assign material, lot and person options. Otherwise, they have to be set prior to beginning the segment.

- Returns

A new MESResponseSegment object.

- Scope

All

Code Examples

Code Snippet

```
operationsResponseUUID = 'c8c77882-9786-4adb-a657-ec4dcc0a4bda'
segmentName = 'Sugar-Nuts Unlimited:Site 1:Area:Line 1'
autoAssignOptions = True
seg=system.mes.createSegmentForOperation
(operationsResponseUUID, segmentName, autoAssignOptions)
print seg
```

Output

```
ResponseSegment (46ab55da-8d4c-4e39-b4dc-61580b8acd08, Sugar-Nu
ts Unlimited:Site 1:Area:Line 1, 0 parents, 0 children, 0
custom properties, 7 complex properties)
```

system.mes.createSublots

Description

Create new material sublots for the provided material lot.

Syntax

system.mes.createSublots(materialLot, sublotCount)

- Parameters

MESMaterialLot materialLot - The MES object to create new material sublots for.

String sublotCount - Number of new material sublots to create in the specified material lot.

- Returns

Material lot object will newly created material sublots as children.

- Scope

All

Code Examples

Code Snippet

```
#This code will create new material sublots for the provided
material lot.
lot = system.mes.loadMESObject('Lot 1111', 'MaterialLot')
sublots = system.mes.createSublots(lot, 2)
```

system.mes.delaySchedule

Description

Delay the operations request by the amount that it is overdue to begin or finish. When this is done, other operations requests may also be delayed if they are in conflict.

Syntax

system.mes.delaySchedule(operationsRequest)

- Parameters

MESOperationsRequest operationsRequest - The MES object to delay the schedule for.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Load an operations request object
operationsRequest = system.mes.loadMESObject('Receive
MaterialA', 'OperationsRequest')
#Delays schedule for the given operations request
system.mes.delaySchedule(operationsRequest)
```

Overview

These script functions are used to delete operation schedule(s).

Method Options

system.mes.deleteSchedule(fromDate, toDate, category)

Description

Delete the list of operation schedules fall into the specified category.

Syntax

system.mes.deleteSchedule(fromDate, toDate, category)

- Parameters

Date fromDate - The date to begin deleting schedules.

Date toDate - The date to stop deleting schedules.

String category - The category of schedules to delete.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Specify the start and end dates
from java.util import Calendar
begin = Calendar.getInstance()
begin.add(Calendar.DAY_OF_MONTH, 1)
start = begin.getTime()
end = Calendar.getInstance()
end.add(Calendar.MONTH, -1)
finish = end.getTime()
#Deletes all 'Active' schedules
system.mes.deleteSchedule(start, finish, 'Active')
```

system.mes.deleteSchedule(operationsScheduleUUID)

Description

Delete the operations schedule with the UUID specified by the operationsScheduleUUID.

Syntax

system.mes.deleteSchedule(operationsScheduleUUID)

- Parameters

String operationsScheduleUUID - The UUID of the operations schedule to delete.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Deletes the schedule of the operation
 system.mes.deleteSchedule('79425a70-4420-4705-a7e4-
f25288898732')
```

system.mes.deriveMESObject

The deriveMESObject allows for an MES Object to be created and have the same properties as the object that is passed to it. This function call be called by passing it an MES Object or by passing it the UUID of an object.

system.mes.deriveMESObject(mesObject, mesObjectTypeName, copyPropertyValues)

Description

Returns a new instance of MES object of the type specified by the mesObjectTypeName parameter and all applicable properties derived from the MES object specified by the mesObject parameter will be copied to it.

- Parameters

AbstractMESObject mesObject - The MES object to derive the new MES object from
String MES Object Type Name - The MES object type name to base the new instance. This can be one of MES object types defined in MESObjectTypes. See MES Object Type Name for more details.
Boolean copyPropertyValues - If true, copy the values of the properties to the new MES object

- Returns

A new instance of the MES Object Type specified in the mesObjectTypeName parameter

- Scope

All

Code Snippet

```
#In the example below, a new MaterialDef MES object is derived.
mesObject = system.mes.loadMESObject('Box', 'MaterialDef')
if mesObject != None:
    system.mes.deriveMESObject(mesObject, 'MaterialDef', True)
```

system.mes.deriveMESObject(mesObjectUUID, mesObjectTypeName, copyPropertyValues)

Description

Returns a new instance of an MES object of the type specified by the mesObjectTypeName parameter and all applicable properties derived from the MES object specified by the mesObjectUUID parameter will be copied to it.

- Parameters

String mesObjectUUID - The UUID of the MES object to base the new object on

String MES Object Type Name - The MES object type name to base the new instance. This can be one of MES object types defined in MESObjectTypes. See MES Object Type Name for more details.

Boolean copyPropertyValues - If true, copy the values of the properties to the the new MES object

- Returns

An AbstractMESObject

- Scope

All

Code Snippet

```
#Returns a new instance of specified MES object
system.mes.deriveMESObject('51b51e30-4d10-41b4-8e0f-
ecde5a5d58f1','MaterialClass', True)
```

system.mes.deriveOperation

Description

Create a new MESOperationsDefinition from the specified operations definition. This also derives all dependent MESOperationsSegment objects.

Syntax

system.mes.deriveOperation(operationsDefinition)

- Parameters

MESOperationsDefinition operationsDefinition - The operations definition to base the derived operations definition.

- Returns

A list containing the newly derived OperationsDefinition and all dependent Operations Segment object. The list is returned as a MES Object List object that is a collection holding MES objects.

- Scope

All

Code Examples

Code Snippet

```
#Create an instance of the given operations definition object
operationsDefinition = system.mes.loadMESObject('Receive
Turkey', 'OperationsDefinition')
system.mes.deriveOperation(operationsDefinition)
```

system.mes.endOperation

Description

End the current operation for the equipment specified by the equipmentPath parameter.

Syntax

system.mes.endOperation(equipmentPath, operationsResponse)

- Parameters

String equipmentPath - Equipment path of the equipment to end the operation.

MESOperationsResponse operationsResponse - The MES object to end. Final properties must be set in the operations response object prior to calling this function. Usually a called to get the current operations response object is done prior to calling this function. Usually, a called to get the current operations response object is done prior to calling this function.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
eqPath = 'My Enterprise\California\Receiving\Unload Station 1'
```

```
operDef = system.mes.loadMESObject('Receive Material', 'Operati
onsDefinition')
#Create an operation based on the operations definition
operResp = system.mes.createOperation(operDef)
system.mes.endOperation(eqPath, operResp)
```

Code Snippet

```
#This code will end the current operation
oper = system.mes.getCurrentOperation('Dressings
Inc\California\Raw Materials\Unload Station 1')
oper.end()
```

Code Snippet

```
#Add new custom property to the operation before it is ended
cp = {'Effort' : ['String', 'Maximum']}
oper.setCustomPropertyValues(cp)
oper.end()
```

system.mes.endSegment

Description

End the specified response segment.

Syntax

system.mes.endSegment(responseSegment)

- Parameters

MESResponseSegment responseSegment - The MES object to end. All final property values must be set prior to ending.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
defSeg = system.mes.loadMESObject('Receive Material', 'Operatio
nsSegment')
eqPath = 'My Enterprise\California\Receiving\Unload Station 1'
respSeg = system.mes.createSegment(defSeg, eqPath)
#Ends the segment
system.mes.endSegment(respSeg)
```

Code Snippet

```
#USE THIS METHOD IF THE SEGMENT'S (END OPERATION WHEN
COMPLETE) SETTING IS TRUE.
seg = system.mes.getActiveSegment('Dressings
Inc\California\Raw Materials\Unload Station 1', 'Unload
Balsamic Vinegar')
seg.end()
```

Code Snippet

```
#USE THIS METHOD IF THE SEGMENT'S (END OPERATION WHEN
COMPLETE) SETTING IS FALSE.
oper = system.mes.getCurrentOperation('Dressings
Inc\California\Raw Materials\Unload Station 1')
seg = oper.getActiveSegment('Unload Balsamic Vinegar')
seg.end()
oper.end()
```

Overview

These script functions are used to execute an MES event.

Method Options

system.mes.executeMESEvent(mesObject, eventName, parameters)

Description

Execute an MES event on the specified MES object with parameters.

Syntax

system.mes.executeMESEvent(mesObject, eventName, parameters)

- Parameters

AbstractMESObject mesObject - The MES object to execute the event on. See AbstractMESObject object in the MES documentation.

String eventName - Name of the event to execute on the specified MES object.

MESObjectEventParameters parameters - Parameters to pass to the event. See MES Object Event Parameters documentation for more information.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Load the MES object
mesObject = system.mes.loadMESObject('VIN 3344', 'MaterialLot')
#Create parameters and execute the event
if mesObject != None:
    params = system.mes.object.parameters.create()
    params.put('Kind', 'Dressing')
    params.put('Priority', 'High')
    system.mes.executeMESEvent(mesObject, 'My User Event',
params)
```

system.mes.executeMESEvent(mesObject, eventName)

Description

Execute an MES event on the specified MES object.

Syntax

system.mes.executeMESEvent(mesObject, eventName)

- Parameters

AbstractMESObject mesObject - The MES object to execute the event on. See AbstractMESObject object in the MES documentation.

String eventName - Name of the event to execute on the specified MES object.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Load the MES object and executes event associated with it
mesObject = system.mes.loadMESObject('Box', 'MaterialDef')
system.mes.executeMESEvent(mesObject, 'Event1')
```

system.mes.executeSegment

Description

Execute the specified response segment. This is the same as begin segment and then ending the segment immediately.

Syntax

system.mes.executeSegment(responseSegment)

- Parameters

MESResponseSegment responseSegment - The MES object to execute. This can be used to split lot, receive material, change lot status, etc.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Load the object
defSeg = system.mes.loadMESObject('Receive Material', 'Operatio
nsSegment')
eqPath = 'My Enterprise\California\Receiving\Unload Station 1'
#Create segment for the MES object
respSeg = system.mes.createSegment(defSeg, eqPath)
system.mes.executeSegment(respSeg)
```

Code Snippet

```
#This code is for the execution of the specified segment
seg = system.mes.createSegment('Unload Vinegar', 'Dressings
Inc\California\Raw Materials\Unload Station 1', False)
seg.setMaterial('Vinegar Type', 'Balsamic Vinegar', 'Dressings
Inc\California\Raw Materials\Tank Farm\Vinegar Tank 1', 'TBV
11000', 100.0)
seg.setPersonnel('Unload Operator', 'Smith, Sam')
seg.execute()
```

Code Snippet

```
#Set custom property value for pH custom property defined in
the segment.
seg.setPropertyValue('pH', 4.4)

#Set custom property value for Brix custom property defined in
the material reference.
cp = {'Brix' : 5.5}
seg.setMaterial('Vinegar', 'Balsamic Vinegar', 'Dressings
Inc\California\Raw Materials\Tank Farm\Vinegar Tank 1', 'TBV
1127', 100.0, cp)
seg.setPersonnel('Operator', 'Hechtman, Tom')
seg.execute()
```

Overview

These script functions are used to execute the specified response segment immediately.

Method Options

system.mes.executeSegmentImmediately(responseSegment, bypassInventoryCheck)

Description

Execute the specified response segment. This is the same as execute segment but bypasses the overhead of the response segment lifetime. In all cases, a new operations response object will be created and associated to the response segment passed as a parameter to this function.

Syntax

system.mes.executeSegmentImmediately(responseSegment, bypassInventoryCheck)

- Parameters

MESResponseSegment responseSegment - The MESResponseSegment object to execute. This can be used to split lot, receive material, change lot status, etc.

Boolean bypassInventoryCheck - If true, bypass the checking of inventory levels.

- Returns

The MESResponseSegment object.

- Scope

All

Code Examples

Code Snippet

```
#Get the current operation
oper = system.mes.getCurrentOperation('[global]\Enterprise\San
Marcos\MP Rotator\MP Rotator 1')
#Create segment for the operation
seg = oper.createSegment('Pack goods')
#Set the material
seg.setMaterial('Molds', 'Mt67b', '[global]\Enterprise\San
Marcos\MP Rotator\MP Rotator 1', 'mld 1127', 100.0)
#Execute immediately
system.mes.executeSegmentImmediately(seg, True)
```

Output

```
ResponseSegment (8f950e94-3bbc-4b10-adea-6f09433c45d7, Pack
goods, 0 parents, 0 children, 0 custom properties, 7 complex
properties)
```

system.mes.executeSegmentImmediately(responseSegment)

Description

Execute the specified response segment. This is the same as execute segment but bypasses the overhead of the response segment lifetime. In all cases, a new operations response object will be created and associated to the response segment passed as a parameter to this function.

Syntax

system.mes.executeSegmentImmediately(responseSegment)

- Parameters

MESResponseSegment responseSegment - The MESResponseSegment object to execute. This can be used to split lot, receive material, change lot status, etc.

- Returns

The MESResponseSegment object.

- Scope

All

Code Examples

Code Snippet

```
#Get current operation
oper=system.mes.getCurrentOperation('[global]\Enterprise\Site
1\Area\Line 1')
#Create segment
seg = oper.createSegment('Mix Nuts')
#Set the material
seg.setMaterial('Mix Peanuts', 'Salt', '[global]
\Enterprise\Site 1\Area\Line 1', 'rt 57', 100)
#Execute immediately
system.mes.executeSegmentImmediately(seg)
```

Output

```
ResponseSegment (0ab4a7c0-99ea-4d79-bed1-cb732388c5a2, Mix
Nuts, 0 parents, 0 children, 0 custom properties, 7 complex
properties)
```

system.mes.exportMESObjects

Description

Convert the specified MES objects to XML. The XML can then be written to a disk file, sent to external systems, written to a database, etc. by using other Ignition script functions.

Syntax

system.mes.exportMESObjects(filter)

- Parameters

MESObject filter filter - A MESObjectFilter object specifying the MES object to include.

- Returns

A XML string value representing all MES objects specified by the filter.

- Scope

All

Code Examples

Code Snippet

```
#This code will print the XML string value.
filter = system.mes.object.filter.createFilter()
filter.setEnableStateName('ENABLED')
filter.setMESObjectTypeName('EquipmentClass')
xml = system.mes.exportMESObjects(filter)
print xml

#This code will write XML string value to a file.
filepath = system.file.openFile("xml")
if filepath != None:
        system.file.writeFile(filepath, xml)
```

Output

```
<?xml version="1.0"?>
<MESObjectList>
    <MESObject MESObjectType="EquipmentClass">
        <CoreProperty name="UUID">a0a7991c-ee75-47d7-8c91-
b0e20e736ea9</CoreProperty>
```

```
        <CoreProperty name="Name">Storage Tank</CoreProperty>
        <CoreProperty name="Description">receiving water<
/CoreProperty>
        <ChildReference>
            <MESObjectType>StorageUnit</MESObjectType>
            <MESObjectUUID>8da06ff8-2922-4e0c-a01a-
e7cda6899a0e</MESObjectUUID>
        </ChildReference>
        <ChildReference>
            <MESObjectType>StorageUnit</MESObjectType>
            <MESObjectUUID>d9eb9a1a-4503-436a-9765-
d57ae8af0ec1</MESObjectUUID>
        </ChildReference>
    </MESObject>
    <MESObject MESObjectType="EquipmentClass">
        <CoreProperty name="UUID">dc161370-e4dd-496c-8d67-
bc306db95843</CoreProperty>
        <CoreProperty name="Name">Substation 1</CoreProperty>
        <CoreProperty name="Description">This is a station to
collect the turkeys</CoreProperty>
    </MESObject>
</MESObjectList>
```

Overview

These script functions are used to get the response segment for the specified equipment path and segment name. This is most often for the purpose of updating the segment with changes in material or personnel.

Method Options

system.mes.getActiveSegment(equipmentPath, segmentName)

Description

Get the response segment for the specified equipment path and segment name.

Syntax

system.mes.getActiveSegment(equipmentPath, segmentName)

- Parameters

String equipmentPath - The path of the equipment that is running an operation and specified segment.

String segmentName - The name of the segment to return.

- Returns

The matching MESResponseSegment object.

- Scope

All

Code Examples

Code Snippet

```
#This code will get the active segment, set the material, and
update the segment to manifest changes.
seg = system.mes.getActiveSegment('Dressings
Inc\California\Raw Materials\Unload Station 1', 'Unload
Balsamic Vinegar')
seg.setMaterial('Vinegar', 'Balsamic Vinegar', 'Dressings
Inc\California\Raw Materials\Tank Farm\Vinegar Tank 2', 'TBV
1128', 100.0)
seg.update()
```

system.mes.getActiveSegment(operationsResponse, equipmentPath, segmentName)

Description

Get the response segment for the specified operation at the equipment path with the given segment name.

Syntax

system.mes.getActiveSegment(operationsResponse, equipmentPath, segmentName)

- Parameters

MESOperationsResponse operationsResponse - The operations response object that is associated with the segment to return.

String equipmentPath - The path of the equipment that is running an operation and specified segment.

String segmentName - The name of the segment to return.

- Returns

The matching MESResponseSegment object.

- Scope

All

Code Examples

Code Snippet

```
operationsResponseUUID = 'a446eea5-d451-4d8f-aac0-60e02672a193'
operationsResponse=system.mes.loadMESObject
(operationsResponseUUID)
segmentName = 'Sugar-Nuts Unlimited:Site 1:Area:Line 1'
path = '[global]\Nuts Unlimited\Site 1\Area\Line 1'
seg=system.mes.getActiveSegment(operationsResponseUUID, path,
segmentName)
print seg
```

Output

```
ResponseSegment (8aff88ce-138e-4bd2-b35d-1a066203bf4c, Sugar-
Nuts Unlimited:Site 1:Area:Line 1, 0 parents, 0 children, 0
custom properties, 7 complex properties)
```

system.mes.getActiveSegment(operationsResponseUUID, equipmentPath, segmentName)

Description

Get the response segment for the operation at the specified equipment path with the given segment name.

Syntax

system.mes.getActiveSegment(operationsResponseUUID, equipmentPath, segmentName)

- Parameters

String operationsResponseUUID - The UUID of the operations response object that is associated with the segment to return.

String equipmentPath - The path of the equipment that is running an operation and specified segment.

String segmentName - The name of the segment to return.

- Returns

The matching MESResponseSegment object.

- Scope

All

Code Examples

Code Snippet

```
operationsResponseUUID = '7a4b2168-1d34-4944-96f5-edc0be02ce32'
path = '[global]\Nuts Unlimited\Site 1\Area\Line 1'
segmentName = 'Sugar-Nuts Unlimited:Site 1:Area:Line 1'
seg=system.mes.getActiveSegment(operationsResponseUUID, path,
segmentName)
seg.end()
```

system.mes.getAvailableOperations

Description

Get a list of the available operations that can be executed on the equipment specified by the equipmentPath parameter.

Syntax

system.mes.getAvailableOperations(equipmentPath, searchPattern, onlyMatchingSegements, onlyProductionVisible)

- Parameters

String equipmentPath - Equipment path of the equipment to return available operations for.

String searchPattern - The search pattern to filter the results by. It can contain the * and ? wild card characters.

Boolean onlyMatchingSegments - If true, only return operations that have a segment with a matching name. This is used to make a single selection instead of having to select and operation and then a segment.

Boolean onlyProductionVisible.

- Returns

A list of available operations. A MESList object is returned that is a collection holding MES object links that represent operations.

- Scope

All

Code Examples

Code Snippet

```
#This will list out all the available operations
oper = system.mes.getAvailableOperations('Dressings
Inc\California\Raw Materials\Unload Station 1', 'Receive *', Tr
ue, True)
```

system.mes.getAvailableReferenceOptions

Description

Get a list of the available options for the specified MES reference property specified in by the property parameter.

Syntax

system.mes.getAvailableReferenceOptions(mesObject, propertyPath, MESObjectTypeName, nameFilter, onlyDefinitionTypes, maxLotReturnCount, lotNameFilter)

- Parameters

AbstractMESObject mesobject -The AbstractMESObject for which the available options.

String propertyPath - The name or property path of the property.

String MES Object Type Name - The type of MES object types to return. See MES Object Type Name for more details.

String nameFilter - A filter that limits the results to the specified type.

Boolean onlyDefinitionTypes - If True, returns only the definition types.

Integer maxLotReturnCount - The maximum number of lots.

String lotNameFilter - The lot name filter used to filter the results.

- Returns

A list of MES Object Link objects holding the options that are appropriate for the specified property. The list is returned as a MES List object that is a collection holding MES object links that represent the options.

- Scope

All

Code Examples

```
Code Snippet
```

```
seg = system.mes.createSegment('Process', '[global]
\Enterprise\Site\Area\Processing Line 1', True)
system.mes.getAvailableReferenceOptions(seg, 'Material.Raw
Material In', 'MaterialLot', '', False, 10, '')
```

system.mes.getAvailableSegments

Description

Get a list of the available segments that can be executed for the specified operation response object.

Syntax

system.mes.getAvailableSegments(operationsResponse , searchPattern)

- Parameters

MESOperationsResponse operationsResponse - The MES object that the segments are defined in.

String searchPattern - The search pattern to filter the results by. It can contain the * and ? wild card characters.

- Returns

A list of available segments. The list is returned as a MESList object that is a collection holding MESObjectLinks for each segment object.

- Scope

All

Code Examples

Code Snippet

```
#This code will print the list of available segments.
```

```
operDef = system.mes.loadMESObject('Receive Material', 'Operati
onsDefinition')
operResp = system.mes.createOperation(operDef)
list = system.mes.getAvailableSegments(operResp, 'Receive *')
for ndx in range(list.size()):
    segments = list.get(ndx)
    print segments
```

Output

```
Receive Material
```

system.mes.getCountValue

Description

Returns the quantity from the automatic production counters.

Syntax

system.mes.getCountValue(equipmentPath, counterName, fromDate, toDate)

- Parameters

String equipmentPath - Equipment path of the equipment to return the current operation response for.

String counterName - The name of the MES counter to return the value for.

Date fromDate - The starting date to base the count value.

Date toDate - The ending date to base the count value.

- Returns

The MES count value.

- Scope

All

Code Examples

Code Snippet

```
from java.util import Calendar

begin = Calendar.getInstance()
begin.add(Calendar.DAY_OF_MONTH, 1)
start = begin.getTime()
end = Calendar.getInstance()
end.add(Calendar.MONTH, -1)
finish = end.getTime()
print system.mes.getCountValue('[global]\My
Enterprise\California\Storage\Vinegar Tanks\Vinegar Tank 1', 'A
BC', start, finish)
```

Output

```
1000
```

system.mes.getCurrentEquipmentStates

Description

Return current equipment states.

Syntax

system.mes.getCurrentEquipmentStates(equipmentPathFilter)

- Parameters

String equipmentPathFilter - The filter value, including * and ? wildcard characters, to filter results by the equipment path.

- Returns

Map<String, EquipmentState> - A map containing the equipment path in the key and the EquipmentState object in the value. See EquipmentState object documentation for details.

- Scope

All

Code Examples

Code Snippet

```
print system.mes.getCurrentEquipmentStates('Enterprise\San
Marcos*')
```

Output

```
 {Enterprise\San Marcos\MP Rotator\MP Rotator 1\Infeed\Nozzle
Assembly=Disabled [RUNNING], Enterprise\San Marcos\MP
Rotator=Unknown State [UNKNOWN], Enterprise\San Marcos\MP
Rotator\MP Rotator 1\2B=Running [RUNNING], Enterprise\San
Marcos\MP Rotator\Test Line 1=Running [RUNNING],
Enterprise\San Marcos\MP Rotator\MP Rotator 1=Non-Production
[IDLE], Enterprise\San Marcos=Unknown State [UNKNOWN],
Enterprise\San Marcos\MP Rotator\Test Line 1\New Cell=Running
[RUNNING], Enterprise\San Marcos\MP Rotator\MP Rotator
1\Infeed=Disabled [RUNNING], Enterprise\San Marcos\MP
Rotator\MP Rotator 1\Infeed\2A=Running [RUNNING],
Enterprise\San Marcos\MP Rotator\MP Rotator 1\Infeed\Nozzle
Assembly\1A=Running [RUNNING], Enterprise\San Marcos\MP
Rotator\MP Rotator 1\Infeed\Nozzle Assembly\1B=Running
[RUNNING]}
```

system.mes.getCurrentOperation

Description

Get the current operations response that is currently running for the equipment specified by the equipmentPath parameter.

Syntax

system.mes.getCurrentOperation(equipmentPath)

- Parameters

String equipmentPath - Equipment path of the equipment to return the current operation response for.

- Returns

The current MESOperationsResponse object for the specified equipment.

- Scope

All

Code Examples

Code Snippet

```
#This code will return the current operation
oper = system.mes.getCurrentOperation('Nuts
Unlimited\Folsom\Packaging\Packaging Line 1')
seg = oper.getActiveSegment('Package Nuts')
```

Output

```
 OperationsResponse (b4532162-ff22-4405-b22a-8436f5c501d3,
Package Nuts, 0 parents, 0 children, 0 custom properties, 1
complex properties)
```

system.mes.getCurrentOperations

Description

Get the current operations response objects that is currently running for the equipment specified by the equipmentPath parameter.

Syntax

system.mes.getCurrentOperations(equipmentPath)

- Parameters

String equipmentPath - Equipment path of the equipment to return the current operations response objects for.

- Returns

A MESObjectList object that holds a collections operations response objects that are currently active for the equipment.

- Scope

All

Code Examples

Code Snippet

```
#specify path for which the current operations are to be listed
eqPath = 'Enterprise\Site\Area\Line 1'
list = system.mes.getCurrentOperations(eqPath)
for i in range(list.size()):
    print list.get(i)
```

Output

```
OperationsResponse (becc61f8-86ca-44fe-9345-626a6ab9151d, PC-
002-Enterprise:Site:Area:Line 1, 0 parents, 0 children, 0
custom properties, 1 complex properties)
```

system.mes.getCurrentSegments

Description

Get the currently executing response segments for the specified equipment.

Syntax

system.mes.getCurrentSegments(equipmentPath)

- Parameters

String equipmentPath - The path of the equipment to return the currently executing response segments.

- Returns

A list of the currently executing response segments for the equipment specified by the equipmentPath parameter. The list is returned as a MESList object that is a collection holding MESObjectLinks for each segment object.

- Scope

All

Code Examples

Code Snippet

```
#This example will return the currently executing segments
segCurrent = system.mes.getCurrentSegments('Dressings
Inc\California\Raw Materials\Unload Station 1')
```

system.mes.getDependencies

Description

Get MES object links of all objects the depend on the specified object.

Syntax

system.mes.getDependencies(mesObject)

- Parameters

AbstractMESObject mesObject - An MES object to return dependencies for. See AbstractMESObject object in the MES documentation.

- Returns

A list of MES Object Link objects that depend on the specified object specified in the mesObjectLink parameter. A MESList object is returned that is a collection holding MES object links.

- Scope

All

Code Examples

Code Snippet

```
#The following is the snippet prints the list of objects.
matClass = system.mes.createMESObject('MaterialClass')
matClass.setPropertyValue('Name', 'Turkey')
matClass.addCustomProperty('Weight', 'Float8', 'Weight of the
turkey', 'Lbs', True, True)
system.mes.saveMESObject(matClass)
results = system.mes.getDependencies(matClass)
```

system.mes.getEquipmentModeHistory

Description

Return the equipment mode history.

Syntax

system.mes.getEquipmentModeHistory(equipmentPath, beginDateTime, endDateTime, includeChildren)

- Parameters

String equipmentPath - The path of equipment to return the mode history for.

Date beginDateTime - The begin date time.

Date endDateTime - The end date time.

String includeChildren - If true, include mode history for children.

- Returns

Dataset - A dataset containing the equipment mode history.

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Enterprise\San Marcos\MP Rotator\Test Line
1'
date = system.date.getDate(2017, 2, 17)
beginDateTime = system.date.setTime(date, 15, 13, 31)
endDateTime = system.date.now()
data=system.mes.getEquipmentModeHistory(eqPath, beginDateTime,
endDateTime, True)
for row in range(data.rowCount):
   for col in range(data.columnCount):
      print data.getValueAt(row, col)
```

Output

```
None
Enterprise\San Marcos\MP Rotator\Test Line 1
Fri Mar 17 15:13:31 PDT 2017
Mon Mar 27 15:10:56 PDT 2017
Production [PRODUCTION]
14397.4166667
0
Enterprise\San Marcos\MP Rotator\Test Line 1\New Cell
Fri Mar 17 15:13:31 PDT 2017
Mon Mar 27 15:10:56 PDT 2017
Production [PRODUCTION]
14397.4166667
```

system.mes.getEquipmentModeOptions

Description

Return the equipment mode options.

Syntax

system.mes.getEquipmentModeOptions(equipmentPath, modeTypeFilter)

- Parameters

String equipmentPath - The path of equipment to return the modes for.

String modeTypeFilter - The equipment mode type filter.

- Returns

MESObjectList - A MESObjectList object containing MESEquipmentMode objects.

- Scope

All

Mode Filters

Valid default values for the modeTypeFilter parameter are...

- 'Unknown'
- 'Production'
- 'Idle'
- 'Changeover'
- 'Maintenance'
- 'Other'
- 'Disabled'

... as well as any custom modes you create.

Code Examples

Code Snippet

```
newData = []
hdr = ['equipPath', 'Name', 'Code', 'Type']


equipPath = '[global]\Nuts Unlimited\Folsom\Receiving\Line 1'

if equipPath != '':
    data = system.mes.getEquipmentModeOptions(equipPath, "")
    for item in data:
        modeName = item.getName()
        modeCode = item.getModeCode()
        modeType = item.getModeTypeName()
        newData.append([equipPath, modeName, modeCode,
modeType])

eqModes = system.dataset.toDataSet(hdr, newData)
for row in range(eqModes.rowCount):
   for col in range(eqModes.columnCount):
      print eqModes.getValueAt(row, col)
```

Output

```
 [global]\Nuts Unlimited\Folsom\Receiving\Line 1
Maintenance
3
Maintenance
[global]\Nuts Unlimited\Folsom\Receiving\Line 1
Changeover
2
Changeover
[global]\Nuts Unlimited\Folsom\Receiving\Line 1
Disabled
0
Disabled
[global]\Nuts Unlimited\Folsom\Receiving\Line 1
Production
1
Production
[global]\Nuts Unlimited\Folsom\Receiving\Line 1
Other
4
Other
```

system.mes.getEquipmentScheduleEntries

Description

Return entries that have been scheduled for an equipment path. The script function returns an MESList object containing MESScheduleEntry objects.

Info

The function requires an additional argument on the Gateway script which is the name of the project.

Syntax

system.mes.getEquipmentScheduleEntries(equipmentPath, beginDate, endDate, categoryFilter, includeProgress)

- Parameters

String equipmentPath - The path for the equipment to return the schedule entries for.

Date beginDate - The beginning date of schedule entries to include in the results.

Date endDate - The ending date of schedule entries to include in the results.

String categoryFilter - The schedule entry categories to include in the results. Multiple schedule categories can be included by separating them with commas.

Boolean includeProgress - If true, each schedule entry will include progress details. Note, this requires more overhead and should only be used if necessary.

- Returns

MESList<MESScheduleEntry> - A list containing MESScheduleEntry objects. Each MESScheduleEntry object contains the links to the operations schedule, operations request, operations response (if one exists) and more schedule details. See MESScheduleEntry in the MES help documentation for more information.

- Scope

All

Code Examples

Code Snippet

```
#specify the equipment path
eqPath = '[global]\Enterprise\Site\Area\Line 1'
#define the start and end dates
begin = system.date.addDays(system.date.now(), -10)
end = system.date.addDays(system.date.now(), 10)

#let category be 'Active'
category = 'Active'

#Gets the equipment schedule entries
list = system.mes.getEquipmentScheduleEntries(eqPath, begin,
end, category, False)
print list
for item in list:
    print item.getScheduledStartDate()
    if item.hasMESOperationsScheduleLink():
        print item.getMESOperationsScheduleLink()
        print item.getMESOperationsRequestLink()

    if item.hasMESOperationsResponseLink():
        print item.getMESOperationsResponseLink()

    print '\n'
```

Output

```
Size 3
2017-04-14 11:23:00.0
(type: Operations Schedule,  uuid: 8689710a-71aa-4c1e-8a9a-
50205d34568f)
PC01-Enterprise:Site:Area:Line 1 ; Operations Request

2017-04-14 12:23:01.0
(type: Operations Schedule,  uuid: d4769a8b-e884-430f-a5c0-
616fbf8201f5)
PC02-Enterprise:Site:Area:Line 1 ; Operations Request

2017-04-18 06:45:07.0
(type: Operations Schedule,  uuid: 0d1d07bc-2da5-4585-8c91-
988c95b36281)
PC01-Enterprise:Site:Area:Line 1 ; Operations Request
```

Overview

These script functions returns equipment state history .

Method Options

system.mes.getEquipmentStateHistory(equipmentPath, beginDateTime, endDateTime, stateTypeFilter, rollupTimeSpan, runLookBackCount)

Description

Return equipment state history.

Syntax

system.mes.getEquipmentStateHistory(equipmentPath, beginDateTime, endDateTime, stateTypeFilter, rollupTimeSpan, runLookBackCount)

- Parameters

String equipmentPath - The path of equipment to return the state history for.

Date beginDateTime - The begin date time.

Date endDateTime - The end date time.

String stateTypeFilter - The equipment state type filter.

Integer rollupTimeSpan - The rollup time span in seconds.

Integer runLookBackCount - The number of runs to return downtime events for within the selected date range. Set to 0 to return all runs within the date range. Set to 1 to return only the current or most recent run.

- Returns

Dataset - A dataset containing the equipment state history.

- Scope

All

Code Examples

Code Snippet

```
equipmentPath = event.source.parent.getComponent('MES Object
Selector').equipmentItemPath
beginDateTime = event.source.parent.getComponent('Date Range').
startDate
endDateTime = event.source.parent.getComponent('Date Range').
endDate
stateTypeFilter = ''
rollupTimeSpan = 0
runLookBackCount = 0
dataset = system.mes.getEquipmentStateHistory(equipmentPath,
beginDateTime, endDateTime, stateTypeFilter, rollupTimeSpan,
runLookBackCount)
print dataset.getColumnNames()
```

Output

```
[Line State Event Begin, Line State Event End, Line State
Duration, Line Downtime Equipment Path, Line Downtime
Equipment Name, Line Downtime Reason, Line Downtime Reason
Path, Equipment Note, Line State Value, Line Downtime State
Time Stamp, Line Downtime Reason Split, Line Downtime Event
Sequence, Line Downtime Occurrence Count, Line State Override
Type, Line State Override Scope]
```

system.mes.getEquipmentStateHistory(equipmentPath, beginDateTime, endDateTime, includeChildren)

Description

Return equipment state history.

Syntax

system.mes.getEquipmentStateHistory(equipmentPath, beginDateTime, endDateTime, includeChildren)

- Parameters

String equipmentPath - The path of equipment to return the state history for.

Date beginDateTime - The begin date time.

Date endDateTime - The end date time.

Boolean includeChildren - If true, include state history for children.

- Returns

Dataset - A dataset containing the equipment state history.

> ⚠ Note that the Line State Object column is an MES Object, and thus this dataset cannot be simply dumped into a table.

- Scope

All

Code Examples

Code Snippet

```
equipmentPath = event.source.parent.getComponent('MES Object
Selector').equipmentItemPath
beginDateTime = event.source.parent.getComponent('Date Range').
startDate
endDateTime = event.source.parent.getComponent('Date Range').
endDate
includeChildren = True
dataset = system.mes.getEquipmentStateHistory(equipmentPath,
beginDateTime, endDateTime, includeChildren)
print dataset.getColumnNames()
```

Output

```
[Equipment Cell Order, Equipment Path, Is Key Cell, State
Begin Time, State End Time, Line State Object, State Duration]
```

system.mes.getEquipmentStateOptions

Description

Return equipment state options.

Syntax

system.mes.getEquipmentStateOptions(equipmentPath, parentUUID, stateTypeFilter)

- Parameters

String equipmentPath - The path of equipment to return the states for.

String parentUUID - The UUID of the parent equipment.

String stateTypeFilter - The equipment state type filter.

- Returns

MESObjectList - A MESObjectList object containing MESEquipmentState or MESEquipmentStateClass objects.

- Scope

All

State Filter

Valid default options for the State filter parameter are...'Unknown'
'Unplanned Downtime'
'Planned Downtime'
'Blocked'
'Starved'
'Running'
'Idle'
'Disabled'
... and any custom states that you have created.

Code Examples

Code Snippet

```
hdr = ['equipPath', 'stateName', 'stateCode', 'stateType']
newData = []


equipPath = '[global]\Nuts Unlimited\Folsom\Receiving\Line 1'

if equipPath != '':
    data = system.mes.getEquipmentStateOptions(equipPath, "", "
")
    for item in data:
        stateName = item.getName()
        if item.getMESObjectType().getName() == 'EquipmentState
Class':
            pass
        else:
            stateCode = item.getStateCode()
            stateType = item.getStateTypeName()
            newData.append([equipPath, stateName, stateCode,
stateType])

eqStates = system.dataset.toDataSet(hdr, newData)
for row in range(eqStates.rowCount):
   for col in range(eqStates.columnCount):
      print eqStates.getValueAt(row, col)
```

Output

```
 [global]\Nuts Unlimited\Folsom\Receiving\Line 1
Unplanned Downtime
3
Unplanned Downtime
[global]\Nuts Unlimited\Folsom\Receiving\Line 1
Planned Downtime
4
Planned Downtime
[global]\Nuts Unlimited\Folsom\Receiving\Line 1
Idle
2
Idle
[global]\Nuts Unlimited\Folsom\Receiving\Line 1
Blocked
5
Blocked
[global]\Nuts Unlimited\Folsom\Receiving\Line 1
Disabled
0
Disabled
```

```
[global]\Nuts Unlimited\Folsom\Receiving\Line 1
Running
1
Running
[global]\Nuts Unlimited\Folsom\Receiving\Line 1
Starved
6
Starved
```

system.mes.getInventory

Description

Get inventory based on the mesLotFilter parameter.

Syntax

system.mes.getInventory(mesLotFilter)

- Parameters

MESLotFilter mesLotFilter - A MESLotFilter object containing the filter criteria to return lot information for. See MESLotFilter object in the MES documentation for more information.

- Returns

Dataset containing the lot results.

- Scope

All

Code Examples

Code Snippet

```
filter = system.mes.lot.filter.createFilter()
filter.setIncludeActiveLots(True)
dataSet = system.mes.getInventory(filter)
pds = system.dataset.toPyDataSet(dataSet)
```

```
value = pds[0][12]
print value
```

Output

```
 Raw Balsamic Vinegar
```

system.mes.getLotInfoByName

Description

Get lot information including custom properties and sublots that belong to the lot.

Syntax

system.mes.getLotInfoByName(lotName, includeSublots, includeCustomProperties)

- Parameters

String lotName - The lot name to return details for.

Boolean includeSublots - If true, include sublot that belong to the lot.

Boolean includeCustomProperties - If true, include custom properties for the lot, and if included, the sublots.

- Returns

A dataset containing a row for each lot. If sublot are included, they reside in a dataset embedded in a column of the lot row. If custom properties are included, they will reside in a dataset embedded in a column of the lot or sublot rows.

- Scope

All

Code Examples

Code Snippet

```
#Get all of the information for material lots with a given name
ds = system.mes.getLotInfoByName('0000000073', False, True)

#Print available columns to access lot information
print 'Available columns to access Material Lot object
information:'
for col in range(ds.columnCount):
    print ds.getColumnName(col)

#Cycle through each one
for row in range(ds.rowCount):

    #The CustomProperties column holds a dataset that has a
row for each custom property
    cpDS = ds.getValueAt(row, 'CustomProperties')

    #This shows the column names of the custom property table
    #These are the available members of the custom property
    print
    print 'Available columns to access custom property
information:'
    for cpCol in range(cpDS.columnCount):
        print cpDS.getColumnName(cpCol)

    print
    print 'Custom property values:'

    #Print name - value of each custom property
    for cpRow in range(cpDS.rowCount):
        name = cpDS.getValueAt(cpRow, 'Name')
        value = cpDS.getValueAt(cpRow, 'Value')
        print '%s = %s' % (name, value)

    #To change custom properties use the following
    lotUUID = ds.getValueAt(row, 'LotUUID')
    seqNo = ds.getValueAt(row, 'LotSequence')
    lotUse = ds.getValueAt(row, 'LotUse')
    if lotUse == 'Out' and seqNo == 1:
        matLot = system.mes.loadMESObject(lotUUID)
        matLot.setPropertyValue('Viscosity', '11000')
        system.mes.saveMESObject(matLot)
```

Output

```
Available columns to access Material Lot object information:
LotUUID
```

```
LotName
LotSequence
LotDescription
LotEnabled
LotAssembly
LotStatus
LotAvailability
LotUnits
MaterialUUID
MaterialName
MaterialDescription
MaterialEnabled
EquipmentUUID
EquipmentName
EquipmentDescription
EquipmentPath
EquipmentEnabled
CustomProperties
Available columns to access custom property information:
MESPropertyUUID
ParentMESPropertyUUID
Name
Description
Value
ValueUnits
ValueDataType
Enable
Required
ProductionVisible
Custom property values:
Available columns to access custom property information:
MESPropertyUUID
ParentMESPropetyUUID
Name
Description
Value
ValueUnits
ValueDataType
Enable
Required
ProductionVisible
Custom property values:
```

system.mes.getLotInfoByUUID

Description

Get lot information including custom properties and sublots that belong to the lot.

Syntax

system.mes.getLotInfoByUUID(lotUUID, includeSublots, includeCustomProperties)

- Parameters

String lotUUID - The lot UUID to return details for.

Boolean includeSublots - If true, include sublot that belong to the lot.

Boolean includeCustomProperties - If true, include custom properties for the lot, and if included, the sublots.

- Returns

A dataset containing a row for each lot. If sublot are included, they reside in a dataset embedded in a column of the lot row. If custom properties are included, they will reside in a dataset embedded in a column of the lot or sublot rows.

- Scope

All

Code Examples

Code Snippet

```
#This Code Snippet will return the details about the Lot
lotInfo = system.mes.getLotInfoByUUID('1a62bcf0-e80d-4319-9efc-6f82409057f6', True, True)
```

Overview

These script functions are used to get all lots currently available at the specified equipment.

Method Options

system.mes.getLotInventoryByEquipment(equipmentUUID, excludeScheduledRequestSegmentUUID)

Description

Get all lots currently available at the specified equipment.

Syntax

system.mes.getLotInventoryByEquipment(equipmentUUID, excludeScheduledRequestSegmentUUID)

- Parameters

String equipmentUUID - The UUID of the equipment to return the available lot details for.

String excludeScheduledRequestSegmentUUID - Optionally, this is a UUID of a request segment to exclude from the results.

- Returns

A list containing information about each lot in the specified equipment. The list is returned as a MESLotQuantitySummaryList object that is a collection holding MESLotQuantitySummaryItem with details for each lot. See MES Lot Quantity Summary List and MES Lot Quantity Summary Item in the MES documentation for more details.

- Scope

All

system.mes.getLotInventoryByEquipment(equipmentPath)

Description

Get all lots currently available at the specified equipment.

Syntax

system.mes.getLotInventoryByEquipment(equipmentPath)

- Parameters

String equipmentPath - Equipment path of the equipment to return the current operation response for.

- Returns

A list containing information about each lot in the specified equipment. The list is returned as a MESLotQuantitySummaryList object that is a collection holding MESLotQuantitySummaryItem with details for each lot. See MES Lot Quantity Summary List and MES Lot Quantity Summary Item in the MES documentation for more details.

- Scope

All

system.mes.getLotInventoryByEquipment(equipmentPath, lotNumberFilter, lotStatusFilter)

Description

Get all lots currently available at the specified equipment.

Syntax

system.mes. getLotInventoryByEquipment(equipmentPath, lotNumberFilter, lotStatusFilter)

- Parameters

String equipmentPath - Equipment path of the equipment to return the current operation response for.

String lotNumberFilter- Custom lot number to filter results.

String lotStatusFilter - Custom lot status value to filter results.

- Returns

A list containing information about each lot in the specified equipment. The list is returned as a MESLotQuantitySummaryList object that is a collection holding MESLotQuantitySummaryItem with details for each lot. See MES Lot Quantity Summary List and MES Lot Quantity Summary Item in the MES documentation for more details.

- Scope

All

Code Examples

Code Snippet

```
lotInventory = system.mes.getLotInventoryByEquipment('[global]
\Turkeys\Folsom\Packaging\Packaging Line 1\Checkweigher', 'V100
0', 'Good')
for lotSummary in lotInventory:
    lotUUID = lotSummary.getMaterialLotUUID()
    lotNo = lotSummary.getLotNumber()
    lotSeq = lotSummary.getLotSequence()
    matUUID = lotSummary.getMaterialUUID()
    matName = lotSummary.getMaterialName()
    matDescription = lotSummary.getMaterialDescription()
    inQuant = lotSummary.getInQuantity()
    outQuant = lotSummary.getOutQuantity()
    schedule = lotSummary.getScheduled()
    available = lotSummary.getAvailable()
    netQuant = lotSummary.getNetQuantity()
    units = lotSummary.getUnits()
    locationLink = lotSummary.getLocationLink()
    print "lot uuid: %s, lot number: %s, lot sequence: %d,
material UUID: %s, material name: %s, material Description: %
s, In Quantity: %f, Out Quantity: %f, schedule: %f, available:
%f, net quantity: %f, units: %s, locationLink: %s" % (lotUUID,
lotNo, lotSeq, matUUID, matName, matDescription, inQuant,
outQuant, schedule, available, netQuant, units, locationLink)
if lotInventory!= None:
        netQuant = lotInventory.getNetQuantitySum()
        inQuant = lotInventory.getInQuantitySum()
        outQuant = lotInventory.getOutQuantitySum()
        schedule = lotInventory.getScheduledSum()
        print " Net Quantity Sum: %f\n In Quantity Sum: %f\n
Out Quantity Sum: %f\n Scheduled Sum:%f" % (netQuant, inQuant,
outQuant, schedule)
```

Output

```
lot uuid: 5afdec7b-c546-4fcb-a5b2-399108966952, lot number: BB
1000, lot sequence: 1, material UUID: 8713506e-b179-4598-a324-b
21d5457a3a8, material name: Butterball Turkey, material
Description: , In Quantity: 0.000000, Out Quantity: 1000.000000
, schedule: 0.000000, available: -1000.000000, net quantity: -1
000.000000, units: None, locationLink: Checkweigher
lot uuid: 9ddeeb1b-23f5-43d6-b9db-c0567134aa12, lot number: BB
1002, lot sequence: 1, material UUID: 8713506e-b179-4598-a324-b
21d5457a3a8, material name: Butterball Turkey, material
```

```
Description: , In Quantity: 100.000000, Out Quantity: 0.000000,
schedule: 0.000000, available: 100.000000, net quantity: 100.00
0000, units: None, locationLink: Checkweigher
lot uuid: 5cd8b1e9-8238-490d-8cbb-108c3960aec6, lot number: BB
1003, lot sequence: 1, material UUID: 8713506e-b179-4598-a324-b
21d5457a3a8, material name: Butterball Turkey, material
Description: , In Quantity: 100.000000, Out Quantity: 0.000000,
schedule: 0.000000, available: 100.000000, net quantity: 100.00
0000, units: None, locationLink: Checkweigher
lot uuid: 72819df8-485d-4abd-9b30-fff1706ccbd5, lot number: BB
1004, lot sequence: 1, material UUID: 8713506e-b179-4598-a324-b
21d5457a3a8, material name: Butterball Turkey, material
Description: , In Quantity: 100.000000, Out Quantity: 0.000000,
schedule: 0.000000, available: 100.000000, net quantity: 100.00
0000, units: None, locationLink: Checkweigher
lot uuid: 0b3a3be9-cc82-4c79-97d4-235f8c1fa79f, lot number: c 3
4fg, lot sequence: 1, material UUID: 3dd45a76-c7dc-449f-99bf-d5
0429e3fb90, material name: Butterball Turkey, material
Description: , In Quantity: 280.000000, Out Quantity: 0.000000,
schedule: 0.000000, available: 280.000000, net quantity: 280.00
0000, units: None, locationLink: Checkweigher
lot uuid: 044708c8-877f-494c-9af3-a36991cac1cf, lot number: BB
1004, lot sequence: 2, material UUID: 3dd45a76-c7dc-449f-99bf-d
50429e3fb90, material name: Butterball Turkey, material
Description: , In Quantity: 100.000000, Out Quantity: 0.000000,
schedule: 0.000000, available: 100.000000, net quantity: 100.00
0000, units: None, locationLink: Checkweigher
lot uuid: 599d2116-2bf8-4348-aede-15cec98bdb6c, lot number: t
bonbbd, lot sequence: 1, material UUID: 3dd45a76-c7dc-449f-99bf
-d50429e3fb90, material name: Butterball Turkey, material
Description: , In Quantity: 122.000000, Out Quantity: 0.000000,
schedule: 0.000000, available: 122.000000, net quantity: 122.00
0000, units: None, locationLink: Checkweigher
lot uuid: 6ab19ac0-2e15-4964-928c-70acc02e23c9, lot number:
Lot 1234, lot sequence: 1, material UUID: 4ba87141-6a9f-489e-ba
2b-49ae835e6a0b, material name: Butterball Turkey, material
Description: , In Quantity: 1000.000000, Out Quantity: 0.000000
, schedule: 0.000000, available: 1000.000000, net quantity: 100
0.000000, units: None, locationLink: Checkweigher
 Net Quantity Sum: 802.000000
 In Quantity Sum: 1802.000000
 Out Quantity Sum: 1000.000000
 Scheduled Sum:0.000000
```

Overview

These script functions are used to get the information about a specific lot.

Method Options

system.mes.getLotInventoryByLot(materialLotUUID, excludeResponseMaterialUUID)

Description

Get all material lot details of the specified lot.

Syntax

system.mes.getLotInventoryByLot(materialLotUUID, excludeResponseMaterialUUID)

- Parameters

String materialLotUUID - The UUID of the material lot object to return the available lot details for.

String excludeResponseMaterialUUID - Optionally, this is a UUID of a response material property to exclude from the results.

- Returns

A list containing information about each lot. The list is returned as a MESLotQuantitySummaryList object that is a collection holding MES Lot Quantity Summary Item with details for each lot. See MES Lot Quantity Summary List and MES Lot Quantity Summary Item in the MES documentation for more details.

- Scope

All

Code Examples

Code Snippet

```
#This Code Snippet returns the information about the Lot
getLotInfo = system.mes.getLotInventoryByLot('8b142f48-f697-
4ee2-82e5-1722208f7818', '5a76a6da-0922-4afd-8ad0-501424dbdac4'
)
```

system.mes.getLotInventoryByLot(lotNumber, sequenceNumber)

Description

Get all material lot details of the specified lot.

Syntax

system.mes.getLotInventoryByLot(lotNumber, sequenceNumber)

- Parameters

String lotNumber - The Number of the material lot object to return the available lot details for.

Integer sequenceNumber - The lot sequence number to return the material lot object for. If it is less than zero, then the lot holding the maximum value is returned and if the same lot number is used for multiple segments, each lot with the same lot number will be assigned a different lot sequence number.

- Returns

A list containing information about each lot. The list is returned as a MES Lot Quantity Summary List object that is a collection holding MES Lot Quantity Summary Item with details for each lot. See MESLotQuantitySummaryList and MES Lot Quantity Summary Item in the MES documentation for more details.

- Scope

All

Code Examples

Code Snippet

```
#This Code Snippet returns the information about the Lot
getLotInfo = system.mes.getLotInventoryByLot('V100', 1)
```

system.mes.getLotInventoryByLot(lotNumber)

Description

Get all material lot details of the specified lot.

Syntax

system.mes.getLotInventoryByLot(lotNumber)

- Parameters

String lotNumber - The Number of the material lot object to return the available lot details for.

- Returns

A list containing information about each lot. The list is returned as a MES Lot Quantity Summary List object that is a collection holding MES Lot Quantity Summary Item with details for each lot. See MESLotQuantitySummaryList and MES Lot Quantity Summary Item in the MES documentation for more details.

- Scope

All

Code Examples

Code Snippet

```
#This Code Snippet returns the information about the Lot
getLotInfo = system.mes.getLotInventoryByLot('V100')
```

system.mes.getLotList

Description

Get list of material lots based on the settings in the mesLotFilter parameter.

Syntax

system.mes.getLotList(mesLotFilter)

- Parameters

MESLotFilter mesLotFilter - The MES object with filter criteria to specify the material lots to return.

- Returns

A list of links representing the matching material lots. The list is returned as a MESList object that is a collection holding MESObjectLinks for each MESMaterialLot object.

- Scope

All

Code Examples

Code Snippet

```
#The following snippet prints the list of links.
from java.util import Calendar

filter = system.mes.lot.filter.createFilter()
filter.setModeName('LOT')
filter.setIncludeInactiveLots(True)
beginCal = Calendar.getInstance()
beginCal.add(Calendar.DAY_OF_MONTH, -30)
filter.setBeginDateTime(beginCal)
endCal = Calendar.getInstance()
filter.setEndDateTime(endCal)
results = system.mes.getLotList(filter)
for link in results:
    print link.getName()
```

Output

```
Lot 1111
Lot 1234
```

MES Lot Filter

Object Description

The MESLotFilter object is used to help when searching for lots or sublots and is required for certain script methods. Lot or sublot search results can be limited by using the MESLotFilter properties to narrow down the MES lots to return when using the system.mes.getLotList script function.

Scripting Functions

The following function can be used to create MESLotFilter.

system.mes.lot.filter.createFilter()

Description

Returns a new instance of a MESLotFilter object for that properties can be set on. This is typically used when a script function requires a MESLotFilter object as a parameter.

Syntax

system.mes.lot.filter.createFilter()

- Parameters

None

- Returns

A new instance of a MESLotFilter object.

Code Snippet

```
from java.util import Calendar
filter = system.mes.lot.filter.createFilter()
filter.setModeName('LOT')
filter.setIncludeInactiveLots(True)
beginCal = Calendar.getInstance()
beginCal.add(Calendar.DAY_OF_MONTH, -30)
filter.setBeginDateTime(beginCal)
endCal = Calendar.getInstance()
filter.setEndDateTime(endCal)
results = system.mes.getLotList(filter)
```

```
for link in results:
    print link.getName()
```

Example

If the MESLotFilter was not provided, then each option would have to be passed as a parameter in the system.mes.getLotList method. There are over a dozen options to filter lot on. This would make it very difficult to use because the line of script would look something like the following:

Code Example 1

```
#Cumbersome method that is NOT used:
system.mes.getLotList('', '000*', '', '', '', '', '', '', '', '
', '', beginDate, endDate,'', '', '')
```

Code Example 2

```
#Instead, using the MESLotFilter object the script look like:
filter = system.mes.lot.filter.createFilter()
filter.setLotNameFilter('000*')
filter.setBeginDateTime(beginDate)
filter.setEndDateTime(endDate)
list = system.mes.getLotList(filter)
```

The second example is the supported method and is much more readable.

Methods

The following methods exist for the MES Lot Filter Object.

getBeginDateTime()

Description

Get the beginning date and time to limit the results to return.

Syntax

getBeginDateTime()

- Parameters

None

- Returns

Calendar The beginning date and time to filter results.

Info

Custom Property Value Filter

Description

The results can be limited to only include items that have a custom property expressions defined by this property that evaluates to true.Example Kind > 3.

getCustomPropertyValueFilter()

Description

Get the list of MESPropertyValueFilter used to filter the results.

Syntax

getCustomPropertyValueFilter()

- Parameters

None

- Returns

List of MES Property Value Filter - The custom property value filter containing information about MESObjectTypes, propertyPath, etc .

getEndDateTime()

Description

Get the ending date and time to limit the results to return.

Syntax

getEndDateTime()

- Parameters

None

- Returns

Calendar The ending date and time to filter results.

getLotEquipmentClassFilter()

Description

Get the lot equipment class filter used to filter the results.

Syntax

getLotEquipmentClassFilter()

- Parameters

None

- Returns

String The lot equipment class filter.

getLotEquipmentNameFilter()

Description

Get the lot equipment name filter used to filter the results.

Syntax

getLotEquipmentNameFilter()

- Parameters

None

- Returns

String The lot equipment name filter.

getLotNameFilter()

Description

Get the lot name filter used to filter the results.

Syntax

getLotNameFilter()

- Parameters

None

- Returns

String The lot name filter.

getLotStatusFilter()

Description

Get the custom lot status of results to return.

Syntax

getLotStatusFilter()

- Parameters

None

- Returns

String - The custom lot status value.

getMaterialClassFilter()

Description

Get the material class filter used to filter the results.

Syntax

getMaterialClassFilter()

- Parameters

None

- Returns

String The material class filter.

getMaterialNameFilter()

Description

Get the material name filter used to filter the results.

Syntax

getMaterialNameFilter()

- Parameters

None

- Returns

String The material name filter.

getMaxResults()

Description

Get the maximum number of results to that will returned.

Syntax

getMaxResults()

- Parameters

None

- Returns

Integer The maximum number of items to return.

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
print filter.getMaxResults()
```

Output

```
100
```

getModeName()

Description

Get the type of results to return. It can be return results for lots (batches) of material or serialized items (sublots).

Syntax

getModeName()

- Parameters

None

- Returns

String Name - The name of the type of results to return.

Code Examples

Code Snippet

```
#Prints the mode names.
filter = system.mes.lot.filter.createFilter()
filter.setIncludeActiveLots(True)
print filter.getModeName()
```

Output

```
LOT
```

getOperationNameFilter()

Description

Get the operation name filter used to filter the results.

Syntax

getOperationNameFilter()

- Parameters

None

- Returns

String The operation name filter.

getPersonnelClassFilter()

Description

Get the personnel class filter used to filter the results.

Syntax

getPersonnelClassFilter()

- Parameters

None

- Returns

String The personnel class filter.

getPersonnelNameFilter()

Description

Get the personnel name filter used to filter the results.

Syntax

getPersonnelNameFilter()

- Parameters

None

- Returns

String The P ersonnelNameFilter.

getSegmentEquipmentClassFilter()

Description

Get the segment equipment class filter used to filter the results.

Syntax

getSegmentEquipmentClassFilter()

- Parameters

None

- Returns

String The segment equipment class filter.

getSegmentEquipmentNameFilter()

Description

Get the segment equipment name filter used to filter the results.

Syntax

getSegmentEquipmentNameFilter()

- Parameters

None

- Returns

String The lot equipment name filter.

getSegmentNameFilter()

Description

Get the segment name filter used to filter the results.

Syntax

getSegmentNameFilter()

- Parameters

None

- Returns

String The segment name filter.

getSublotNameFilter()

Description

Get the sublot name filter used to filter the results.

Syntax

getSublotNameFilter()

- Parameters

None

- Returns

String The sublot name filter.

hasCustomPropertyValuefilter()

Description

Checks to see if a custom property value filter exists for the given lot.

Syntax

hasCustomPropertyValuefilter()

- Parameters

None

- Returns

True, if there exist a custom property value filter .

- Scope

All

includeActiveLots()

Description

If True, lots or sublots currently being processed will be included in the results.

Syntax

includeActiveLots()

- Parameters

None

- Returns

Boolean If True, active lots will be return in the results.

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.includeInactiveLots()
```

includeInactiveLots()

Description

If True, lots or sublots that are complete will be included in the results.

Syntax

includeInactiveLots()

- Parameters

None

- Returns

Boolean - If True, completed lots will be return in the results.

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.includeInactiveLots()
```

setBeginDateTime(beginDateTime)

Description

Set the beginning date and time to limit results to return. This applies to lots or sublots that are completed or have a custom lot status.

Syntax

setBeginDateTime(beginDateTime)

- Parameters

Calendar beginDateTime - Beginning date and time to filter results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
from java.util import Calendar
filter = system.mes.lot.filter.createFilter()
begin = Calendar.getInstance()
begin.add(Calendar.MONTH, -1)
filter.setBeginDateTime(begin)
```

setCustomPropertyValueFilter(customPropertyValueFilter)

Description

Set the custom property filter expressions to filter the results. If a custom property of a MES object matches an expression in this list, then it will be included in the results. Use system.mes.object.filter.parseCustomPropertyValueFilter() script function to create the list of MES Property Value Filter objects.

Syntax

setCustomPropertyValueFilter(customPropertyValueFilter)

- Parameters

List of MES Property Value Filter customPropertyValueFilter - The custom property value list to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
filter = system.mes.object.filter.createFilter()
list = system.mes.object.filter.
parseCustomPropertyValueFilter('pH > 5.0,Width = 2.5')
filter.setCustomPropertyValueFilter(list)
```

setEndDateTime(endDateTime)

Description

Set the ending date and time to limit results to return. This applies to lots or sublots that are completed or have a custom lot status.

Syntax

setEndDateTime(endDateTime)

- Parameters

Calendar endDateTime - Ending date and time to filter results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example using the current time as the endTime
from java.util import Calendar
filter = system.mes.lot.filter.createFilter()
endTime = Calendar.getInstance()
filter.setEndDateTime(endTime)
```

setIncludeActiveLots(includeActiveLots)

Description

If set to True, lots or sublots that are actively being processed will be included in the results.

Syntax

setIncludeActiveLots(includeActiveLots)

- Parameters

Boolean includeActiveLots - If True, include active lots or sublots in results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setIncludeActiveLots(True)
```

setIncludeInactiveLots(includeInactiveLots)

Description

If set to True, lots or sublot that are completed will be included in the results.

Syntax

setIncludeInactiveLots(includeInactiveLots)

- Parameters

Boolean includeActiveLots - If True, include completed lots or sublots in results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setIncludeInactiveLots(True)
```

setLotEquipmentClassFilter(lotEquipmentClassFilter)

Description

Set the lot equipment class filter to include lots that were stored in the equipment that belong to the equipment class that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setLotEquipmentClassFilter(lotEquipmentClassFilter)

- Parameters

String lotEquipmentClassFilter - The lot equipment class filter used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
```

```
filter = system.mes.lot.filter.createFilter()
filter.setLotEquipmentClassFilter('Storage Tank')
```

setLotEquipmentNameFilter(lotEquipmentNameFilter)

Description

Set the lot equipment name filter to include lots that were stored in the equipment with a name that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setLotEquipmentNameFilter(lotEquipmentNameFilter)

- Parameters

String lotEquipmentNameFilter - The lot equipment name filter used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setLotEquipmentNameFilter('Vinegar Tank?')
```

setLotNameFilter(lotNameFilter)

Description

Set the lot name to filter to include in the results. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setLotNameFilter(lotNameFilter)

- Parameters

String lotNameFilter - The lot name filter used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setLotNameFilter('V100*')
```

setLotStatusFilter(lotStatusFilter)

Description

Set the custom lot status of results to return. If the Final Lot Status property in a resource definition of a Process Segment or Operations Segment is set to a custom lot status, it can be filtered with this property.

Syntax

setLotStatusFilter(lotStatusFilter)

- Parameters

String lotStatusFilter - Custom lot status value to filter results.

- Returns

Nothing

Code Examples

```
Code Snippet

#Example
filter = system.mes.lot.filter.createFilter()
filter.setLotStatusFilter('Complete')
```

setMaterialClassFilter(materialClassFilter)

Description

Set the material class filter to include lots that have material that belong to the material class that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setMaterialClassFilter(materialClassFilter)

- Parameters

String materialClassFilter - The material class filter used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setMaterialClassFilter('* Vinegar')
```

setMaterialNameFilter(materialNameFilter)

Description

Set the material definition name filter to include lots that have material with a name that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setMaterialNameFilter(materialNameFilter)

- Parameters

String materialNameFilter - The material definition name filter used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setMaterialNameFilter('* Turkey')
```

setMaxResults(maxResults)

Description

Set the maximum results to return. This prevents a large list from being returned which reduces database operations, memory usage and other resources when most of the time, the results are not used. If large results are needed, then this property can be increased.

Syntax

setMaxResults(maxResults)

- Parameters

Integer maxResults - The maximum number of items to return.

- Returns

Nothing

Code Examples

Code Snippet

```
#Here is an example of how to set the maximum result count.
filter = system.mes.lot.filter.createFilter()
filter.setMaxResults(200)
print filter.getMaxResults()
```

Output

```
200
```

setModeName(modeName)

Description

Set the type of results to return. It can be return results for lots (batches) of material or serialized items (sublots). Options are Lot and Sublot.

Syntax

setModeName(modeName)

- Parameters

String modeName - The name of the mode for the type of results to return.

- Returns

Nothing

Code Examples

Code Snippet

```
#This code snippet will set the mode name.
filter = system.mes.lot.filter.createFilter()
filter.setModeName('Sublot')
```

setOperationNameFilter(operationNameFilter)

Description

Set the operation name filter to include lots that were processed by the operation with a name that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setOperationNameFilter(operationNameFilter)

- Parameters

String operationNameFilter - The operation name filter used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setOperationNameFilter('Receive*')
```

setPersonnelClassFilter(personnelClassFilter)

Description

Set the personnel class filter to include lots that were processed by personnel that belong to the personnel class that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setPersonnelClassFilter(personnelClassFilter)

- Parameters

String personnelClassFilter - The personnel class filter used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setPersonnelClassFilter('Operator?')
```

setPersonnelNameFilter(personnelNameFilter)

Description

Set the personnel name filter to include lots that were processed with a name that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setPersonnelNameFilter(personnelNameFilter)

- Parameters

String personnelNameFilter - The personnel name filter used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setPersonnelNameFilter('Jo*')
```

setSegmentEquipmentClassFilter(segmentEquipmentClassFilter)

Description

Set the segment equipment class filter to include lots that were processed at the equipment that belong to the equipment class that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setSegmentEquipmentClassFilter(segmentEquipmentClassFilter)

- Parameters

String segmentEquipmentClassFilter - The segment equipment class filter used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setSegmentEquipmentClassFilter('* Tank')
```

setSegmentEquipmentNameFilter(segmentEquipmentNameFilter)

Description

Set the segment equipment name filter to include lots that were processed at the equipment with a name that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setSegmentEquipmentNameFilter(segmentEquipmentNameFilter)

- Parameters

String segmentEquipmentNameFilter - The segment equipment name filter used to filter the results.

- Returns

Nothing

Code Examples

```
Code Snippet

#Example
filter = system.mes.lot.filter.createFilter()
filter.setSegmentEquipmentNameFilter('Vinegar*')
```

setSegmentNameFilter(segmentNameFilter)

Description

Set the segment name filter to include lots that were processed by the segment with a name that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setSegmentNameFilter(segmentNameFilter)

- Parameters

String segmentNameFilter - The segment name filter used to filter the results.

- Returns

Nothing

Code Examples

```
Code Snippet
```

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setSegmentNameFilter('Receive*')
```

setSublotNameFilter(sublotNameFilter)

Description

Set the sublot name to filter to include in the results. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setSublotNameFilter(sublotNameFilter)

- Parameters

String sublotNameFilter - The sublot name filter used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Example
filter = system.mes.lot.filter.createFilter()
filter.setSublotNameFilter('BB 100?')
```

Overview

These script functions are used to get lot trace details currently available at the specified lot name.

Method Options

system.mes.getLotTraceByLotName(lotName, highlightSublotName)

Description

Get lot trace details for the specified lot name.

Syntax

system.mes.getLotTraceByLotName(lotName, highlightSublotName)

- Parameters

String lotName - The lot name to return trace information for.

String highlightSublotName - Optionally, include the name of the sublot to highlight. If this parameter is not blank, an additional column will be added and set to 1 if the specified sublot is contained in a lot.

- Returns

A dataset containing the trace results.

- Scope

All

Code Examples

Code Snippet

```
#This code returns the trace details
getTrace = system.mes.getLotTraceByLotName('V 1000', 'SN9823')
```

system.mes.getLotTraceByLotName(lotName, highlightSublotName, maxFanCount, detailedMode)

Description

Get lot trace details for the specified lot name.

Syntax

system.mes.getLotTraceByLotName(lotName, highlightSublotName, maxFanCount, detailedMode)

- Parameters

String lotName - The lot name to return trace information for.

String highlightSublotName - Optionally, include the name of the sublot to highlight. If this parameter is not blank, an additional column will be added and set to 1 if the specified sublot is contained in a lot.

Integer maxFanCount - The maximum number of lots to fan out to before returning an error. Compiling huge amounts of trace data requires system resources and typically is not useful to the end user. This setting is a safety to prevent inadvertently requesting huge amounts of trace data.

Boolean detailedMode - If true, more details are returned in the trace results.

- Returns

A dataset containing the trace results.

- Scope

All

Code Examples

Code Snippet

```
#This code returns the trace details
system.mes.getLotTraceByLotName('V 1000', 'SN9823', 100, True)
```

Overview

These script functions are used to get lot trace details currently available at the specified lot UUID.

Method Options

system.mes.getLotTraceByLotUUID(lotUUID, highlightSublotName)

Description

Get lot trace details for the specified lot UUID.

Syntax

system.mes.getLotTraceByLotUUID(lotUUID, highlightSublotName)

- Parameters

String lotUUID - The lot UUID to return trace information for.

String highlightSublotName - Optionally, include the name of the sublot to highlight. If this parameter is not blank, an additional column will be added and set to 1 if the specified sublot is contained in a lot.

- Returns

A dataset containing the trace results.

- Scope

All

Code Examples

Code Snippet

```
#This code returns the trace information of the Lot
getTrace = system.mes.getLotTraceByLotUUID('ff6dc96a-0968-4ae2-8127-ef22bb9cbc02', 'SN9823')
```

system.mes.getLotTraceByLotUUID(lotUUID, highlightSublotName, maxFanCount, detailedMode)

Description

Get lot trace details for the specified lot UUID.

Syntax

system.mes.getLotTraceByLotUUID(lotUUID, highlightSublotName, maxFanCount, detailedMode)

- Parameters

String lotUUID - The lot UUID to return trace information for.

String highlightSublotName - Optionally, include the name of the sublot to highlight. If this parameter is not blank, an additional column will be added and set to 1 if the specified sublot is contained in a lot.

Integer maxFanCount - The maximum number of lots to fan out to before returning an error. Compiling huge amounts of trace data requires system resources and typically is not useful to the end user. This setting is a safety to prevent inadvertently requesting huge amounts of trace data.

Boolean detailedMode - If true, more details are returned in the trace results.

- Returns

A dataset containing the trace results.

- Scope

All

Code Examples

Code Snippet

```
#This code returns the trace information of the Lot
getTrace = system.mes.getLotTraceByLotUUID('ff6dc96a-0968-4ae2-8127-ef22bb9cbc02', 'SN9823', 100, True)
```

Overview

These script functions are used to get lot trace details currently available at the specified sublot name.

Method Options

system.mes.getLotTraceBySublotName(sublotName)

Description

Get lot trace details for the specified sublot Name.

Syntax

system.mes.getLotTraceBySublotName(sublotName)

- Parameters

String sublotName - The sublot name to return trace information for.

- Returns

A dataset containing the trace results.

- Scope

All

Code Examples

Code Snippet

```
#This snippet is used to return the lot trace details
new = system.mes.getLotTraceBySublotName('SN9823')
```

system.mes.getLotTraceBySublotName(sublotName, maxFanCount, detailedMode)

Description

Get lot trace details for the specified sublot Name.

Syntax

system.mes.getLotTraceBySublotName(sublotName, maxFanCount, detailedMode)

- Parameters

String sublotName - The sublot name to return trace information for.

Integer maxFanCount - The maximum number of lots to fan out to before returning an error. Compiling huge amounts of trace data requires system resources and typically is not useful to the end user. This setting is a safety to prevent inadvertently requesting huge amounts of trace data.

Boolean detailedMode - If true, more details are returned in the trace results.

- Returns

A dataset containing the trace results.

- Scope

All

Code Examples

Code Snippet

```
#This snippet is used to return the lot trace details
new = system.mes.getLotTraceBySublotName('SN9823', 100, True)
```

Overview

These script functions are used to get lot trace details currently available at the specified sublot name.

Method Options

system.mes.getLotTraceBySublotUUID(sublotUUID)

Description

Get lot trace details for the specified sublot UUID.

Syntax

system.mes.getLotTraceBySublotUUID(sublotUUID)

- Parameters

String sublotUUID - The UUID of the sublot to return trace information for.

- Returns

A dataset containing the trace results.

- Scope

All

Code Examples

Code Snippet

```
#This Code will return the dataset containing the trace results
getTrace = system.mes.getLotTraceBySublotUUID('1a62bcf0-e80d-
4319-9efc-6f82409057f6')
```

system.mes.getLotTraceBySublotUUID(sublotUUID, maxFanCount, detailedMode)

Description

Get lot trace details for the specified sublot UUID.

Syntax

system.mes.getLotTraceBySublotUUID(sublotUUID, maxFanCount, detailedMode)

- Parameters

String sublotUUID - The UUID of the sublot to return trace information for.

Integer maxFanCount - The maximum number of lots to fan out to before returning an error. Compiling huge amounts of trace data requires system resources and typically is not useful to the end user. This setting is a safety to prevent inadvertently requesting huge amounts of trace data.

Boolean detailedMode - If true, more details are returned in the trace results.

- Returns

A dataset containing the trace results.

- Scope

All

Code Examples

Code Snippet

```
#This Code will return the dataset containing the trace results
getTrace = system.mes.getLotTraceBySublotUUID('1a62bcf0-e80d-
4319-9efc-6f82409057f6', 100, True)
```

system.mes.getMaterialLotNextAvailable

Description

Get a link the next available material lot object for the specified segment and material property.

Syntax

system.mes.getMaterialLotNextAvailable(responseSegment, materialPropertyName, lotNamePattern)

- Parameters

MESResponseSegment responseSegment - The mes object used when determining the next available lot.

String materialPropertyName - The name of the material property to get next available lot.

String lotNamePattern - Optionally, this is a pattern to filter the lot names by when determining the next available lot. Default lotNamePatterns are EquipmentName and LotNumber.

- Returns

A MES Object Link object representing to the next available lot.

- Scope

All

Code Examples

Code Snippet

```
#This code will print the name of next available lot.
responseSeg = system.mes.loadMESObject('8b4a6fc1-20ef-4417-
a0fe-97d2987bf83b')
print system.mes.getMaterialLotNextAvailable(responseSeg, 'New
Material', 'EquipmentName')
```

Output

```
Vinegar Tank 1
```

system.mes.getMESAnalysisSettingsList

Description

Return a list of names of stored analysis settings.

Syntax

system.mes.getMESAnalysisSettingsList()

- Parameters

None

- Returns

List<String> - A list object containing strings of the stored analysis names.

- Scope

All

Code Examples

Code Snippet

```
system.mes.getMESAnalysisSettingsList()
```

Output

```
[report]
```

system.mes.getMESDemoSupport

Description

Get the support from the MES Demo project.

Syntax

system.mes.getMESDemoSupport()

- Parameters

None

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
system.mes.getMESDemoSupport()
```

system.mes.getMESObjectChildLinks

Description

Get the children of an MES object that is specified by the filter parameter.

Syntax

system.mes.getMESObjectChildLinks(filter)

- Parameters

MESObjectFilter filter - The MES object containing criteria to select MES object to return the children for.

- Returns

A list of MES Object Link objects containing the children of the specified MES object.

- Scope

All

Code Examples

Code Snippet

```
filter = system.mes.object.filter.createFilter()
filter.setEnableStateName('ENABLED')
filter.setMESObjectTypeName('Equipment')
filter.setMESObjectNamePattern('Unload *')
system.mes.getMESObjectChildLinks(filter)
```

MES Object Filter

Base Object

The MES Object Filter is derived from the MESAbstractObject and inherits all the exposed properties, methods and events for that object.

Scripting Functions

- system.mes.object.filter.createFilter() can be used to create an **MESObjectFilter** object
- system.mes.object.filter.parseCustomPropertyValueFilter(filter) takes in an **MESObjectFilter** object as a parameter

Example: Creating an MES Object Filter

```
myMESObjectFilter = system.mes.object.filter.createFilter()
print myMESObjectFilter.getMESObjectTypeName()
```

Object Description

The **MESObjectFilter** object is used to specify the MES objects to return for various components such as the MES Object Selector and script functions.

Methods

Beside the common MESAbstractObject methods, the following methods exist for the MES Object Filter.

getCustomPropertyNamePattern()

Description

Get the custom property name pattern used to filter the results.

Syntax

getCustomPropertyNamePattern()

- Parameters

None

- Returns

String The custom property name pattern.

getCustomPropertyValueFilter()

Description

Get the list of MESPropertyValueFilter used to filter the results.

Syntax

getCustomPropertyValueFilter()

- Parameters

None

- Returns

List of MESPropertyValueFilter - The custom property value list.

getEnabledStateName()

Description

Get the enable state to filter the results.

Syntax

getEnabledStateName()

- Parameters

None

- Returns

String name - The name of the enable state.

getMESObjectNamePattern()

Description

Get the MES object name pattern used to filter the results.

Syntax

getMESObjectNamePattern()

- Parameters

None

- Returns

String The MES object name pattern.

getMESObjectTypeName()

Description

Return the MES object type name the filter is set for.

Syntax

getMESObjectTypeName()

- Parameters

None

- Returns

String The MES object type name.

getMESObjectTypes()

Description

Gets the MES object types associated with the specified MES object filter.

Syntax

getMESObjectTypes()

- Parameters

None

- Returns

MESObjectTypes - A list of mes object types associated with this filter.

- Scope

All

getMESObjectUUIDList()

Description

Returns a list of MES object UUIDs to return in the results.

Syntax

getMESObjectUUIDList()

- Parameters

None

- Returns

List of String - A list of MES object UUIDs.

Code Snippet

```
filter = system.mes.object.filter.createFilter()
#Gets the list of uuid.
list = filter.getMESObjectUUIDList()
Addind another item to the list.
list.add('5253ccae-47b4-4dc2-954f-900ffa8636eb')
```

getPrimaryClassFilter()

Description

Gets the primary class filter that has been set.

Syntax

getPrimaryClassFilter()

- Parameters

None

- Returns

The primary class filter which was previously defined to filter the results.

getPrimaryMESObjectPath()

Description

Gets the primary MES object path that was set to filter the results.

Syntax

getPrimaryMESObjectPath()

- Parameters

None

- Returns

The primary MES object path to filter the results.

getPrimaryMESObjectUUID()

Description

Get the UUID of the primary MES object to include in the results.

Syntax

getPrimaryMESObjectUUID()

- Parameters

None

- Returns

String The primary MES object UUID.

hasCustomPropertyNamePattern()

Description

Checks for the existence of a custom property name pattern.

Syntax

hasCustomPropertyNamePattern()

- Parameters

None

- Returns

boolean - True, if there exist a custom property name pattern and False otherwise.

- Scope

All

hasCustomPropertyValueFilter()

Description

Checks if there is a custom property value to filter the results.

Syntax

hasCustomPropertyValueFilter()

- Parameters

None

- Returns

boolean - True, if there exist a custom property value filter and False otherwise.

- Scope

All

hasMESObjectNamePattern()

Description

Checks if there is an MES object name pattern to filter the results.

Syntax

hasMESObjectNamePattern()

- Parameters

None

- Returns

boolean - True, if there exist an MES object name pattern and False otherwise.

- Scope

All

hasMESObjectTypes()

Description

Checks whether there is any MES object type name to filter the results.

Syntax

hasMESObjectTypes()

- Parameters

None

- Returns

boolean - True, if there exist any MES object type defined to filter the results and False otherwise.

- Scope

All

hasMESObjectUUIDs()

Description

Checks if there is MES object uuids to filter the results.

Syntax

hasMESObjectUUIDs()

- Parameters

None

- Returns

boolean - True, if there exist some uuids to filter the results.

hasPrimaryClassFilter()

Description

Checks if there is any primary class filter associated with this MES object filter.

Syntax

hasPrimaryClassFilter()

- Parameters

None

- Returns

boolean - True, if there exist a primary class filter and False otherwise.

hasPrimaryMESObjectPath()

Description

Checks whether there is a primary MES object path to filter the results.

Syntax

hasPrimaryMESObjectPath()

- Parameters

None

- Returns

boolean - True, if there exist a primary MES object path and False otherwise.

hasPrimaryMESObjectUUID()

Description

Checks for the existence of primary MES object uuid to filter the results.

Syntax

hasPrimaryMESObjectUUID()

- Parameters

None

- Returns

boolean - True, if there exist a primary MES object uuid and False otherwise.

isIncludeRelated()

Description

The results can be limited to only include the related items that is defined by this property that evaluates to true.

Syntax

isIncludeRelated()

- Parameters

None

- Returns

boolean - True if the MES object include related items and False otherwise.

setCustomPropertyNamePattern(customPropertyNamePattern)

Description

Set the custom property name pattern to filter the results. If a MES object contains a custom property that matches the custom property name pattern, then it will be included in the results.

Syntax

setCustomPropertyNamePattern(customPropertyNamePattern)

- Parameters

String customPropertyNamePattern - The custom property name pattern used to filter the results.

- Returns

Nothing

Code Examples

```
Code Snippet

filter = system.mes.object.filter.createFilter()
filter.setCustomPropertyNamePattern('Type')
```

setCustomPropertyValueFilter(customPropertyValueFilter)

Description

Set the custom property filter expressions to filter the results. If a custom property of a MES object matches an expression in this list, then it will be included in the results. Use system.mes.object.filter.parseCustomPropertyValueFilter() script function to create the list of MESPropertyValueFilter objects.

Syntax

setCustomPropertyValueFilter(customPropertyValueFilter)

- Parameters

List of MES Property Value Filter customPropertyValueFilter - The custom property value list to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Create a filter.
filter = system.mes.object.filter.createFilter()
#Parses the expression and returns a list of
MESPropertyValueFilter objects that are used in filters.
list = system.mes.object.filter.
parseCustomPropertyValueFilter('pH > 5.0,Width = 2.5')
filter.setCustomPropertyValueFilter(list)
```

Code Examples

Code Snippet

```
filter = system.mes.object.filter.createFilter()
list = system.mes.object.filter.
parseCustomPropertyValueFilter('Item Number=A12SIK')
filter.setCustomPropertyValueFilter(list)
list = system.mes.searchMESObjects(filter)
for ndx in range(list.size()):
    mesObject = list.get(ndx)
    print mesObject.getName()
```

Output

```
84001
```

setEnableStateName(name)

Description

Set the enable state to filter the results.

Options:
DISABLED
ENABLED
BOTH

Syntax

setEnableStateName(name)

- Parameters

String name - The name of the enable state.

- Returns

Nothing

Code Examples

Code Snippet

```
#Here's how to set the enable state.
filter = system.mes.object.filter.createFilter()
filter.setEnableStateName('Disabled')
```

setIncludeRelated(includeRelated)

Description

Sets the include related property to filter the results.

Syntax

setIncludeRelated(includeRelated)

- Parameters

boolean includeRelated - Set this to True, if results should only include related objects and set to False otherwise.

- Returns

Nothing

setMESObjectNamePattern(mesObjectNamePattern)

Description

Set the MES object name pattern to include in the results. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

Syntax

setMESObjectNamePattern(mesObjectNamePattern)

- Parameters

String mesObjectNamePattern - The MES object name pattern used to filter the results.

- Returns

Nothing

Code Examples

Code Snippet

```
#Create a filter.
filter = system.mes.object.filter.createFilter()
#Here is an example for setting the name pattern.
filter.setMESObjectNamePattern('*Turkey')
list = system.mes.searchMESObjects(filter)
```

```
for ndx in range(list.size()):
    mesObject = list.get(ndx)
    print mesObject.getMESObjectType().getDisplayName()
```

Output

```
Response Material Definition
Material Definition
Response Material Class
Response Material Class
Material Class
```

setMESObjectTypeName(mesObjectTypeNames)

Description

Set the MES object type name to filter the results.

Syntax

setMESObjectTypeName(mesObjectTypeNames)

- Parameters

String name - Name of the MES object type to limit the results. Options are: EquipmentClass, Equipment, Enterprise, Site, Area, Line, LineCell, LineCellGroup, StorageZone, StorageUnit.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
filter = system.mes.object.filter.createFilter()
#Name of MESObjectType is set to "EquipmentClass."
filter.setMESObjectTypeName('EquipmentClass')
```

setMESObjectTypes(mesObjectTypes)

Description

Sets the MES object types to filter the results.

Syntax

setMESObjectTypes(mesObjectTypes)

- Parameters

MESObjectTypes - The MES object types to set as filter.

- Returns

Nothing

- Scope

All

setMESObjectUUIDList(mesObjectUUIDList)

Description

Set the UUIDs of the MES objects to return in the results.

Syntax

setMESObjectUUIDList(mesObjectUUIDList)

- Parameters

List of String mesObjectUUIDList - The list of UUIDs to include in the results.

- Returns

Nothing

setPrimaryClassFilter(primaryClassFilter)

Description

The results can be limited to only include items that have a primary class filter defined by this property that evaluates to true.

Syntax

setPrimaryClassFilter(primaryClassFilter)

- Parameters

String primaryClassFilter - The primary class to filter the results.

- Returns

Nothing

setPrimaryMESObjectPath(primaryMESObjectPath)

Description

Set the path of the primary MES object to include in the results.

Syntax

setPrimaryMESObjectPath(primaryMESObjectPath)

- Parameters

String primaryMESObjectPath - The path of the primary MES object to include the results.

- Returns

Nothing

setPrimaryMESObjectUUID(primaryMESObjectUUID)

Description

Set the UUID of the primary MES object to include in the results. Child MES objects will also be included in the results.

Syntax

setPrimaryMESObjectUUID(primaryMESObjectUUID)

- Parameters

String primaryMESObjectUUID - The UUID of the primary MES object to include the results.

- Returns

Nothing

Code Examples

Code Snippet

```
filter = system.mes.object.filter.createFilter()
filter.setPrimaryMESObjectUUID('73facb39-806c-4bfc-8881-
cc06707a9909')
```

Properties

The following properties are available for this object.

- None

Overview

These script functions are used to retrieve MES object links.

Method Options

system.mes.getMESObjectLink(mesObjectTypeName, mesObjectUUID)

Description

Get a MESObjectLink for the specified by the name of the MES object type and UUID.

Syntax

system.mes.getMESObjectLink(MESObjectTypeName, mesObjectUUID)

- Parameters

String MES Object Type Name - The name of MES object type of the link to return. See MES Object Type Name for more details.

String mesObjectUUID - The UUID of the MES object to return the MES object link for.

- Returns

A MES Object Link for the specified MES object type and UUID. The MES object type will be looked up in the MES object cache or database. A MESObjectLink contains basic information about the MES object without the overhead of the the full MES object.

- Scope

All

Code Examples

Code Snippet

```
print system.mes.getMESObjectLink('EquipmentClass', 'a0a7991c-ee75-47d7-8c91-b0e20e736ea9')
```

Output

```
Storage Tank
```

system.mes.getMESObjectLink(mesObjectUUID)

Description

Get a MESObjectLink for the specified UUID.

Syntax

system.mes.getMESObjectLink(mesObjectUUID)

- Parameters

String mesObjectUUID - The UUID of the MES object to return the MES object link for.

- Returns

A MESObjectLink for the specified MES object type and UUID. The MES object type will be looked up in the MES object cache or database. A MESObjectLink contains basic information about the MES object without the overhead of the the full MES object.

- Scope

All

Code Examples

Code Snippet

```
print system.mes.getMESObjectLink(''a8a3771b-6330-423b-89fb-b54b7933fe3e')
```

Output

```
Receive Steel
```

system.mes.getMESObjectLink(mesObjectType, mesObjectUUID)

Description

Get a MESObjectLink for the specified MES object type and UUID.

Syntax

system.mes.getMESObjectLink(MESObjectType, mesObjectUUID)

- Parameters

String mesObjectType - The MES object type name of the MES object link to return.

String mesObjectUUID - The UUID of the MES object to return the MES object link for.

- Returns

A MES Object Link for the specified MES object type and UUID. The MES object type will be looked up in the MES object cache or database. A MESObjectLink contains basic information about the MES object without the overhead of the full MES object.

- Scope

All

Code Examples

Code Snippet

```
print system.mes.getMESObjectLink('OperationsDefinition', 'a8a3
771b-6330-423b-89fb-b54b7933fe3e')
```

Output

```
Receive Steel
```

system.mes.getMESObjectLinkByEquipmentPath

Description

Get a MESObjectLink for the specified equipment path.

Syntax

system.mes.getMESObjectLinkByEquipmentPath(equipmentPath)

- Parameters

String equipmentPath - The equipment path to return the MESObjectLink for the associated MES object.

- Returns

A MES Object Link object. It can be a link to one of the following MES object types MESEnterprise, MESSite, MESArea, MESLine, MESLineCell, MESLineCellGroup, MESStorageZone or MESStorageUnit. A MESObjectLink contains basic information about the MES object without the overhead of the full MES object.

- Scope

All

Code Examples

Code Snippet

```
#Get the Tank 1A equipment object from the equipment path
eqLink = system.mes.getMESObjectLinkByEquipmentPath('[global]
\Dressings Inc\California\Raw Materials\Tank Farm\Tank 1A')
```

system.mes.getMESObjectLinkByName

Description

Get a MES Object Link for the name of the MES object specified.

Syntax

system.mes.getMESObjectLinkByName(mesObjectTypeName, mesObjectName)

- Parameters

String MES Object Type Name - The MES object type to base the new instance. This can be one of MES object types defined in MESObjectTypes. See MES Object Type Name for more details.

String mesObjectName - Name of the MES object to get the link for.

- Returns

MES Object Link object specified by the name of the MES object.

- Scope

All

Code Examples

Code Snippet

```
clsLink = system.mes.getMESObjectLinkByName('EquipmentClass', '
Unload Stations')
refLinkList = system.mes.getReferencedMESObjects(clsLink)
for ndx in range(refLinkList.size()):
    print refLinkList.get(ndx)
```

system.mes.getNextMESObjectName

Description

Return the next available name. If the proposed name is already used, the next available sequential number will be appended to the end.

Syntax

system.mes.getNextMESObjectName(mesObjectTypeName, mesObjectUUID, name)

- Parameters

String MES Object Type Name - The MES object type to generate the next name for. Because names must be unique amount the category of MES object type, this is required. See MES Object Type Name for more details.

String mesObjectUUID - When this script function is called for an existing MES object, this is the UUID for it so that it will ignore duplicate name check with itself. Otherwise, just pass an empty string.

String name - Proposed name to use.

- Returns

Next available name.

- Scope

All

Code Examples

Code Snippet

```
#This Snippet is an example that retrieves information about
the next available name.
  system.mes.getNextMESObjectName('StorageUnit', '8da06ff8-
2922-4e0c-a01a-e7cda6899a0e', 'StorageZone')
```

system.mes.getOperationProductionCount

Description

Return the production count for an operations response. The track production by setting must be set to a valid material reference.

Syntax

system.mes.getOperationProductionCount(operationsResponseUUID)

- Parameters

String operationsResponseUUID - The UUID of the operations response MES object to return the production count for.

- Returns

Double count - The production count as a double (Float8).

- Scope

All

Code Examples

Code Snippet

```
system.mes.getOperationProductionCount('fb613d7a-e8d0-40b8-8fde-d84294444002')
```

Output

```
9.0
```

system.mes.getOperationSegments

Description

Get the segments that belong to the specified operations response. Note, for available segments that an active operations response can run, use getAvailableSegments() instead.

Syntax

system.mes.getOperationSegments(operationsLink, searchPattern)

- Parameters

MESObjectLink operationsLink - The MES object link to an operation definition, operations version, operations request or operations response object to return the associated segments.

String searchPattern - The search pattern to filter the results by. It can contain the * and ? wild card characters.

- Returns

A list of links representing all segments of the operation. The list is returned as a MESList object that is a collection holding MESObjectLinks for each segment object.

- Scope

All

Code Examples

Code Snippet

```
objLink = system.mes.object.link.create('OperationsDefinition',
'ec8fa61c-2dce-4499-870a-04ee6e778c15')
system.mes.getOperationSegments(objLink, 'Receive *')
```

system.mes.getProductionItemByEquipmentPath

Description

Returns the production item specified by the equipment path.

Syntax

system.mes.getProductionItemByEquipmentPath(equipmentPath)

- Parameters

String equipmentPath - The path of the equipment to return the production item for.

- Returns

The production item specified by the equipment path.

- Scope

All

Code Examples

Code Snippet

```
eqPath='[global]\Nuts Unlimited\Folsom\Mixing\Mixing Line 1'
system.mes.getProductionItemByEquipmentPath(eqPath)
```

Output

```
(Mixing Line 1, 105415f9-e2ea-4b86-925e-6c606a5856c0)
```

system.mes.getReferencedMESObjects

Description

Gets the list of reference objects specified by the definitionMESLink parameter.

Syntax

system.mes.getReferencedMESObjects(definitionMESLink)

- Parameters

MES Object Link definitionMESLink - The MES object link as to which the referenced objects are returned for.

- Returns

MESList - The list is returned as a MESList object that is a collection holding MESObjectLinks for each reference object.

- Scope

All

Code Examples

```
Code Snippet

clsLink = system.mes.getMESObjectLinkByName('EquipmentClass', '
Unload Stations')
refLinkList = system.mes.getReferencedMESObjects(clsLink)
for ndx in range(refLinkList.size()):
    print refLinkList.get(ndx)
```

system.mes.getScheduleOperations

Description

Get a list of MES object links for each operations request for the operations schedule specified by the operationsScheduleUUID parameter.

Syntax

system.mes.getScheduleOperations(operationsScheduleUUID)

- Parameters

String operationsScheduleUUID - The UUID of the operations schedule to return the associated operations requests for.

- Returns

A list containing links for each operations request. The list is returned as a MESList object that is a collection holding MESObjectLinks for each operations request object.

- Scope

All

Code Examples

Code Snippet

```
#This code will print the name of the operationsRequest object
with the specified ScheduleRefUUID.
operationsSchedule = system.mes.getScheduleOperations('72b408bf
-aaf4-420f-9ec5-4aa7fabbff83')
for request in operationsSchedule :
    print request
```

Output

```
Receive Turkeys
```

system.mes.getSublotInfoByName

Description

Get sublot information including custom properties.

Syntax

system.mes.getSublotInfoByName(sublotName, includeCustomProperties)

- Parameters

String sublotName - The sublot name to return details for.

Boolean includeCustomProperties - If true, include custom properties for the sublot.

- Returns

A dataset containing a row for each sublot. If custom properties are included, they will reside in a dataset embedded in a column of the sublot row.

- Scope

All

Code Examples

Code Snippet

```
#This Code Snippet will return a dataset containing
information about the sublot
lotInfo = system.mes.getSublotInfoByName('SN9823', True)
```

system.mes.getSublotInfoByUUID

Description

Get sublot information including custom properties.

Syntax

system.mes.getSublotInfoByUUID(sublotUUID, includeCustomProperties)

- Parameters

String sublotUUID - The UUID for the sublot to return details for.

Boolean includeCustomProperties - If true, include custom properties for the sublot.

- Returns

A dataset containing a row for each sublot. If custom properties are included, they will reside in a dataset embedded in a column of the sublot row.

- Scope

All

Code Examples

Code Snippet

```
#This Code Snippet will return a dataset containing
information about the sublot
sublot = system.mes.getSublotInfoByUUID('896e25aa-671a-416d-
bf5e-7fa377812a55', True)
```

system.mes.getTagCollectorDeltaValue

Description

Return the different between the values specified by the beginDateTime and endDateTime. Only MES tag collectors that record numeric values support this functionality.

⚠ This script function works only with the **Equipment Count** tag collector type.

Syntax

system.mes.getTagCollectorDeltaValue(equipmentPath, collectorType, key, beginDateTime, endDateTime)

- Parameters

String equipmentPath - The path from the production model to the desired equipment.

String collectorType - The name of the tag collector type.

String key - Where there are multiple i nstances of a tag collector type, this specifies which one to use . For example, there can be multiple MES counters for the "Equipment Count" tag collector type. If not needed, pass an empty string. In other words it is the name of the MES counter or the name of the Additional Factor.

Date beginDateTime - The starting date to include in the returned values.

Date endDateTime - The ending date to include in the returned values.

- Returns

Object deltaValue - The difference between the value at the endDateTime and the value at beginDateTime.

- Scope

All

Code Examples

Code Snippet

```
#The following code will display the tag  collector delta
value in a numeric label component
#Copy this code snippet to actionPerformed event handler of a
button

eqPath = event.source.parent.getComponent('MES Object Selector'
).equipmentItemPath
collectorType = "Equipment Count"
key = "Material Out"

#Get the begin and end date from two Popup Calendar components
fromDate = event.source.parent.getComponent('Popup Calendar').
date
toDate = event.source.parent.getComponent('Popup Calendar 1').
date

value = system.mes.getTagCollectorDeltaValue(eqPath,
collectorType, key, fromDate, toDate)

event.source.parent.getComponent('Numeric Label').value = value
```

Output

```
4
```

system.mes.getTagCollectorLastTimeStamp

Description

Returns the timestamp of the last value chronologically recorded by this MES tag collector.

⚠ This function is not supported for the 'Equipment Count' collector type

Syntax

system.mes.getTagCollectorLastTimeStamp(equipmentPath, collectorType, key)

- Parameters

String equipmentPath - The path from the production model to the desired equipment.

String collectorType - The name of the tag collector type. See Tag Collector Types for more details.

String key - Where there are multiple i nstances of a tag collector type, this specifies which one to use . For example, there can be multiple MES counters for the "Equipment Count" tag collector type. If not needed, pass an empty string. In other words it is the name of the MES counter or the name of the Additional Factor.

- Returns

Date lastTimeStamp - The timestamp of the last chronological recorded value.

- Scope

All

Code Examples

Code Snippet

```
dateTime = system.date.now()
equipmentPath = '[global]\Enterprise\San Marcos\MP Rotator\MP
Rotator 1'
collectorType = 'Equipment Mode'
key = ''

system.mes.getTagCollectorLastTimeStamp(equipmentPath,
collectorType, key)
```

Output

```
 Mon Mar 20 16:29:31 PDT 2017
```

system.mes.getTagCollectorLastValue

Description

Return the last value chronologically recorded by this MES tag collector.

Syntax

system.mes.getTagCollectorLastValue(equipmentPath, collectorType, key)

- Parameters

String equipmentPath - The path from the production model to the desired equipment.

String collectorType - The name of the tag collector type. See Tag Collector Types for more details.

String key - Where there are multiple i nstances of a tag collector type, this specifies which one to use . For example, there can be multiple MES counters for the "Equipment Count" tag collector type. If not needed, pass an empty string. In other words it is the name of the MES counter or the name of the Additional Factor.

- Returns

Object value - The last chronological recorded value.

- Scope

All

Code Examples

Code Snippet

```
equipmentPath = equipmentPath = '[global]\\Enterprise\\San
Marcos\\MP Rotator\\MP Rotator 1'
collectorType = 'Equipment State'
key = ''
print system.mes.getTagCollectorLastValue(equipmentPath,
collectorType, key)
```

```
Output

 89
```

system.mes.getTagCollectorPreviousTimeStamp

Description

Return the timestamp of the value just previous to the specified date and time.

Syntax

system.mes.getTagCollectorPreviousTimeStamp(equipmentPath, collectorType, key, dateTime)

- Parameters

String equipmentPath - The path from the production model to the desired equipment.

String collectorType - The name of the tag collector type. See Tag Collector Types for more details.

String key - Where there are multiple i nstances of a tag collector type, this specifies which one to use . For example, there can be multiple MES counters for the "Equipment Count" tag collector type. If not needed, pass an empty string. In other words it is the name of the MES counter or the name of the Additional Factor.

Date dateTime - The date to start searching for the previous timestamp to return.

- Returns

Date previousTimeStamp - The timestamp of the value just prior to the specified dateTime value.

- Scope

All

Code Examples

Code Snippet

```
#Prints the previous time stamp
dateTime = system.date.now()
equipmentPath = '[global]\Enterprise\San Marcos\MP Rotator\MP
Rotator 1'
collectorType = 'Equipment State'
key = ''

print system.mes.getTagCollectorPreviousTimeStamp
(equipmentPath, collectorType, key, dateTime)
```

Output

```
2017-03-20 15:13:31.0
```

system.mes.getTagCollectorPreviousValue

Description

Return the value just previous to the specified date and time.

Syntax

system.mes.getTagCollectorPreviousValue(equipmentPath, collectorType, key, dateTime)

- Parameters

String equipmentPath - The path from the production model to the desired equipment.

String collectorType - The name of the tag collector type. See Tag Collector Types for more details.

String key - Where there are multiple i nstances of a tag collector type, this specifies which one to use . For example, there can be multiple MES counters for the "Equipment Count" tag collector type. If not needed, pass an empty string. In other words it is the name of the MES counter or the name of the Additional Factor.

Date dateTime - The date to start searching for the previous value to return.

- Returns

Object value - The value just prior to the specified date and time.

- Scope

All

Code Examples

Code Snippet

```
dateTime = system.date.now()
equipmentPath = '[global]\Enterprise\San Marcos\MP Rotator\MP
Rotator 1'
collectorType = 'Equipment Mode'
key = ''

print system.mes.getTagCollectorPreviousValue(equipmentPath,
collectorType, key, dateTime)
```

Output

```
 3
```

Overview

These script functions are used to return a single value that has previously been recorded for the MES tag collector.

The Equipment State tag collector has the following auxiliary values.

- EquipmentUUID - The unique identifier for the equipment.
- State - The current equipment state.
- OriginalState - The original equipment state before it was updated.
- DifferedToUUID - If the original EquipmentUUID is changed using the Downtime Table then the new uuid is DifferedToUUID.
- DifferedState - If the original state is changed using the Downtime Table then the new state is DifferedState.

Method Options

system.mes.getTagCollectorValue(equipmentPath, collectorType, key, dateTime)

Description

Return a single value that has previously been recorded for the MES tag collector.

Syntax

system.mes.getTagCollectorValue(equipmentPath, collectorType, key, dateTime)

- Parameters

String equipmentPath - The path from the production model to the desired equipment.

String collectorType - The name of the tag collector type. See Tag Collector Types for more details.

String key - Where there are multiple i nstances of a tag collector type, this specifies which one to use . For example, there can be multiple MES counters for the "Equipment Count" tag collector type. If not needed, pass an empty string. In other words it is the name of the MES counter or the name of the Additional Factor.

Date dateTime - The date of the value to return.

- Returns

Object value - The single value recorded for the specified dateTime.

- Scope

All

Code Examples

Code Snippet

```
date = system.date.getDate(2017, 2, 20)
dateTime = system.date.setTime(date, 15, 13, 31)
equipmentPath = '[global]\Enterprise\San Marcos\MP Rotator\MP
Rotator 1'
```

```
collectorType = 'Equipment Mode'
key = ''
tagValue = system.mes.getTagCollectorValue(equipmentPath,
collectorType, key, dateTime)
print tagValue
```

Output

```
4
```

system.mes.getTagCollectorValue(equipmentPath, collectorType, auxValueName, key, dateTime)

Description

Return a single value that has previously been recorded for the MES tag collector.

Syntax

system.mes.getTagCollectorValue(equipmentPath, collectorType, auxValueName, key, dateTime)

- Parameters

String equipmentPath - The path from the production model to the desired equipment.

String collectorType - The name of the tag collector type. See Tag Collector Types for more details.

String auxValueName - This specifies which auxiliary value to update. For example, the Equipment State tag collector has an "OriginalState" auxiliary value. See Auxiliary value for more information.

String key - Where there are multiple i nstances of a tag collector type, this specifies which one to use . For example, there can be multiple MES counters for the "Equipment Count" tag collector type. If not needed, pass an empty string. In other words it is the name of the MES counter or the name of the Additional Factor.

Date dateTime - The date of the value to return.

- Returns

Object value - The single value recorded for the specified dateTime.

- Scope

All

Code Examples

Code Snippet

```
date = system.date.getDate(2017, 2, 20)
dateTime = system.date.setTime(date, 15, 13, 31)
equipmentPath = '[global]\Enterprise\San Marcos\MP Rotator\MP
Rotator 1'
collectorType = 'Equipment State'
key = ''
tagValue = system.mes.getTagCollectorValue(equipmentPath,
collectorType, 'OriginalState', key, dateTime)
print tagValue
```

Output

```
 8
```

system.mes.getTagCollectorValues

Description

Return MES tag collector values for a given date range.

Syntax

system.mes.getTagCollectorValues(equipmentPath, collectorType, key, beginDateTime, endDateTime)

- Parameters

String equipmentPath - The path from the production model to the desired equipment.

String collectorType - The name of the tag collector type. See Tag Collector Types for more details.

String key - Where there are multiple i nstances of a tag collector type, this specifies which one to use . For example, there can be multiple MES counters for the "Equipment Count" tag collector type. If not needed, pass an empty string. In other words it is the name of the MES counter or the name of the Additional Factor.

Date beginDateTime - The starting date of the values returned.

Date endDateTime - The ending date of the values returned.

- Returns

Dataset values - A Dataset containing columns for the timestamps and the values withing the specified date range.

- Scope

All

Code Examples

Code Snippet

```
date = system.date.getDate(2017, 2, 20)
beginDateTime = system.date.setTime(date, 15, 13, 31)
endDateTime = system.date.now()
equipmentPath = '[global]\Enterprise\San Marcos\MP Rotator\MP
Rotator 1'
collectorType = 'Equipment Mode'
key = ''
data = system.mes.getTagCollectorValues(equipmentPath,
collectorType, key, beginDateTime, endDateTime)
for row in range(data.rowCount):
   for col in range(data.columnCount):
      print data.getValueAt(row, col)
```

Output

```
Mon Mar 20 15:13:31 PDT 2017
2
```

```
Mon Mar 20 15:13:31 PDT 2017
3
```

system.mes.hasDependencies

Description

Determine if a MES object has other MES objects that rely on it.

Syntax

system.mes.hasDependencies(mesObject)

- Parameters

AbstractMESObject mesObject - An MES object to check if dependencies exists. See AbstractMESObject object in the MES documentation.

- Returns

True, if other MES object(s) depend on the specified object.

- Scope

All

Code Examples

Code Snippet

```
#This code will check the dependency
matClass = system.mes.createMESObject('MaterialClass')
matClass.setPropertyValue('Name', 'Turkey')
matClass.addCustomProperty('Weight', 'Float8', 'Weight of the
turkey', 'Lbs', True, True)
system.mes.saveMESObject(matClass)
system.mes.hasDependencies(matClass)
```

system.mes.importMESObjects

Description

Imports MES Objects specified by the XML string provided.

Syntax

system.mes.importMESObjects(xml)

- Parameters

String xml - An XML string value representing all MES objects to import.

- Returns

A list of MES objects derived from the XML.

- Scope

All

Code Examples

Sample Input

```
<?xml version="1.0"?>
<MESObjectList>
    <MESObject MESObjectType="AnalysisSettings">
        <CoreProperty name="UUID">028167c1-c8d5-4fcf-bafb-
1107acb9b498</CoreProperty>
        <CoreProperty name="Name">ImportTest</CoreProperty>
        <CoreProperty name="Enabled">true</CoreProperty>
        <CoreProperty name="Creator">Unknown</CoreProperty>
        <CoreProperty name="OwnerUserName">admin</CoreProperty>
        <CoreProperty name="IsPublic">true</CoreProperty>
        <CoreProperty name="DataPoints">Infeed-Material In,OEE
Infeed Count,Standard Count,OEE Performance</CoreProperty>
        <CoreProperty name="Filter">Equipment Path =
'Enterprise\Site\Area\Line 1'</CoreProperty>
```

```
        <CoreProperty name="IncludeDrillDownOptions">true</Core
Property>
        <CoreProperty name="SettingValues">Last Values=True</Co
reProperty>
        <ComplexProperty kind="AnalysisSecurity" name="Administ
rator" uuid="765f70b0-911b-4146-9db2-35e259a1d8dd">
            <ComplexMember name="CanExecute">true</ComplexMembe
r>
            <ComplexMember name="CanModify">true</ComplexMember
>
        </ComplexProperty>
    </MESObject>
</MESObjectList>
```

Code Snippet

```
##This code will read the XML file and MES Objects listed in
it that are imported.
##Each object must be saved to be made manifest in the
system.  This can be done
##individually or using the saveMESObjects() function with an
MES Object List input.
path = system.file.openFile("xml")
if path != None:
    xml = system.file.readFileAsString(path)
    MESObjectList = system.mes.importMESObjects(xml)
    system.mes.saveMESObjects(MESObjectList)
```

system.mes.invalidateCache

Description

Clears the MES Object cache on the gateway causing MES Objects to be reloaded from the database. For performance purposes, this script function should be used sparingly.

Syntax

system.mes.invalidateCache()

- Parameters

None

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet

#This code removes the cache from the gateway
clear = system.mes.invalidateCache()
```

system.mes.loadDisabledMESObject

Description

Load and returns a disabled MES object.

Syntax

system.mes.loadDisabledMESObject(name, mesObjectTypeName)

- Parameters

String name - The name of the MES object.

String MES Object Type Name - The name of the type of MES object. See MES Object Type Name for more details.

- Returns

An Abstract mes object. (See AbstractMESObject object in the MES documentation).

- Scope

All

Code Examples

```
Code Snippet

 system.mes.loadDisabledMESObject('Storage Tank', 'EquipmentCla
ss')
```

Overview

These script functions are used to load a specific material lot.

Method Options

system.mes.loadMaterialLot(lotNumber, sequenceNumber, equipmentPath, onlyAvailableLot)

Description

Get a specified material lot object.

Syntax

system.mes.loadMaterialLot(lotNumber, sequenceNumber, equipmentPath, onlyAvailableLot)

- Parameters

String lotNumber - The lot number to return the material lot object for.

Integer sequenceNumber - The lot sequence number to return the material lot object for. If it is less than zero, then the lot holding the maximum value is returned and if the same lot number is used for multiple segments, each lot with the same lot number will be assigned a different lot sequence number.

String equipmentPath - The equipmentPath to the MES object.

Boolean onlyAvailableLot - If true, then only the available lot will be returned.

- Returns

A MESMaterialLot object.

- Scope

All

Code Examples

```
Code Snippet

#This code returns the specified material lot.
system.mes.loadMaterialLot('Lot 1111', 1, 'My
Enterprise\California\Storage\Vinegar Tanks\Vinegar Tank 1', Tr
ue)
```

system.mes.loadMaterialLot(lotNumber, sequenceNumber, onlyAvailableLot)

Description

Get a specified material lot object.

Syntax

system.mes.loadMaterialLot(lotNumber, sequenceNumber, onlyAvailableLot)

- Parameters

String lotNumber - The lot number to return the material lot object for.

Integer sequenceNumber - The lot sequence number to return the material lot object for. If it is less than zero, then the lot holding the maximum value is returned and if the same lot number is used for multiple segments, each lot with the same lot number will be assigned a different lot sequence number.

Boolean onlyAvailableLot - If true, then only the available lot will be returned.

- Returns

A MESMaterialLot object.

- Scope

All

Code Examples

Code Snippet

```
#This code returns the specified material lot.
 system.mes.loadMaterialLot('Lot 1111', 1, True)
```

Overview

These script functions are used to load the link to a specific material lot.

Method Options

system.mes.loadMaterialLotLink(lotNumber, sequenceNumber, equipmentPath, onlyAvailableLot)

Description

Get a link to the specified material lot object.

Syntax

system.mes.loadMaterialLotLink(lotNumber, sequenceNumber, equipmentPath, onlyAvailableLot)

- Parameters

String lotNumber - The lot number to return the link for.

Integer sequenceNumber - The lot sequence number to return the link for. If it is less than zero, then the lot holding the maximum value is returned and if the same lot number is used for multiple segments, each lot with the same lot number will be assigned a different lot sequence number.

String equipmentPath - The equipmentPath to the MES object.

Boolean onlyAvailableLot - If true, then only the available lot will be returned.

- Returns

A MES Object Link object representing the material lot.

- Scope

All

Code Examples

Code Snippet

```
#This code returns the link to the specified material lot.
system.mes.loadMaterialLotLink('Lot 1111', 1, 'My
Enterprise\California\Storage\Vinegar Tanks\Vinegar Tank 1', Tr
ue)
```

system.mes.loadMaterialLotLink(lotNumber, sequenceNumber, onlyAvailableLot)

Description

Get a link to the specified material lot object.

Syntax

system.mes.loadMaterialLotLink(lotNumber, sequenceNumber, onlyAvailableLot)

- Parameters

String lotNumber - The lot number to return the link for.

Integer sequenceNumber - The lot sequence number to return the link for. If it is less than zero, then the lot holding the maximum value is returned and if the same lot number is used for multiple segments, each lot with the same lot number will be assigned a different lot sequence number.

Boolean onlyAvailableLot - If true, then only the available lot will be returned.

- Returns

A MES Object Link object representing the material lot.

- Scope

All

Code Examples

Code Snippet

```
#This code returns the link to the specified material lot.
system.mes.loadMaterialLotLink('Lot 1111', 1, True)
```

Overview

These script functions are used to load the MES object.

Method Options

loadMESObject(mesObjectUUID)

Description

Load and returns a MES object based on the mesObjectUUID parameter.

Syntax

system.mes.loadMESObject(mesObjectUUID)

- Parameters

String mesObjectUUID - The UUID of the MES object which is its unique ID.

- Returns

An AbstractMESObject object (See AbstractMESObject object in the MES documentation).

- Scope

All

Code Examples

Code Snippet

```
#Get the MES object for a given uuid.
obj = system.mes.loadMESObject('ff6dc96a-0968-4ae2-8127-
ef22bb9cbc02')
```

loadMESObject(name, mesObjectTypeName)

Description

Load and returns MES object based on the name and mesObjectTypeName parameters.

Syntax

system.mes.loadMESObject(name, mesObjectTypeName)

- Parameters

String name - The name of the MES object.

String MES Object Type Name - The name of the type of MES object. See MES Object Type Name for more details.

- Returns

An AbstractMESObject object (See AbstractMESObject object in the MES documentation).

- Scope

All

Code Examples

Code Snippet

```
#Get the MES object for a given name and MES object type.
obj = system.mes.loadMESObject('Box', 'MaterialDef')
```

system.mes.loadMESObjectByEquipmentPath

Description

Load and returns an MES object based on the equipmentPath parameter.

Syntax

system.mes.loadMESObjectByEquipmentPath(equipmentPath)

- Parameters

String equipmentPath - The path of equipment object to load.

- Returns

An AbstractMESObject object. It can be a MESEnterprise, MESSite, MESArea, MESLine, MESLineCell, MESLineCellGroup, MESStorageZone or MESStorageUnit type of MES object.

- Scope

All

Code Examples

Code Snippet

```
obj = system.mes.loadMESObjectByEquipmentPath('[global]\My
Enterprise\California\Storage\Vinegar Tanks\Vinegar Tank 1')
print obj.getName()
```

Output

```
Vinegar Tank 1
```

system.mes.loadMESObjects

Description

Returns a list MES objects based on the filters specified in the filter parameter.

Syntax

system.mes.loadMESObjects(filter)

- Parameters

MESObjectfilter filter - A filter that limits the AbstractMESObject objects to return. (See MES Object Filter object in the MES documentation).

- Returns

A list of MES objects that match the specified filter.

- Scope

All

Code Examples

Code Snippet

```
#This code will print the list of objects
filter = system.mes.object.filter.createFilter()
list = system.mes.object.filter.parseCustomPropertyValueFilter(
'pH > 5.0,Width = 2.5')
filter.setCustomPropertyValueFilter(list)
filter.setMESObjectNamePattern('* Turkey*')
results = system.mes.loadMESObjects(filter)
for link in results:
    print link.getName()
```

Output

```
Butterball Turkey
```

```
Receive Turkeys
Move Turkeys
```

system.mes.loadSchedule

Description

Load and return a list of MES objects associated with the operation schedule specified by the operationsScheduleUUID parameter.

Syntax

system.mes.loadSchedule(operationsScheduleUUID)

- Parameters

String operationsScheduleUUID - The UUID of the operations schedule object to load.

- Returns

A list containing the operations schedule and all associated operations requests and request segments. The list is returned as a MESObjectList object that is a collection holding MES objects.

- Scope

All

Code Examples

Code Snippet

```
#This example would pass operationsScheduleUUID into the
loadSchedule and prints out the object.

objList = system.mes.loadSchedule('ca6ecd0b-d78d-40d4-9e89-
41b5cf8e3f6b')
for ndx in range(objList.size()):
    obj = objList.get(ndx)
    print obj
```

Output

```
OperationsSchedule (915b97de-da40-47c8-a72f-ca41996ebac0, Auto
Test Schedule, 0 parents, 0 children, 0 custom properties, 1 co
mplex properties)
OperationsRequest (73fbb46c-7b98-45b2-9f32-9314e039c7e7, Auto
Test, 0 parents, 0 children, 0 custom properties, 2 complex
properties)
RequestSegment (83262dd0-b343-4859-98f0-a7f2f59e4ffd, Auto
Test, 0 parents, 0 children, 0 custom properties, 6 complex
properties)
```

system.mes.lot.filter.createFilter

Description

Returns a new instance of a MESLotFilter object for that properties can be set on. This is typically used when a script function requires a MESLotFilter object as a parameter.

Syntax

system.mes.lot.filter.createFilter()

- Parameters

None

- Returns

A new instance of a MES Lot Filter object.

Code Snippet

```
from java.util import Calendar

filter = system.mes.lot.filter.createFilter()
filter.setModeName('LOT')
filter.setIncludeInactiveLots(True)
beginCal = Calendar.getInstance()
beginCal.add(Calendar.DAY_OF_MONTH, -30)
```

```
filter.setBeginDateTime(beginCal)
endCal = Calendar.getInstance()
filter.setEndDateTime(endCal)
results = system.mes.getLotList(filter)
for link in results:
    print link.getName()
```

system.mes.notifyEquipmentDataChanged

Description

When this is called, the MES analysis engine will check for new data from the last time stamp in the cache to the current time.

Syntax

system.mes.notifyEquipmentDataChanged(equipmentPath)

- Parameters

String equipmentPath - The required path to the equipment that cache will check for new data.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
eqPath='[global]\Nuts Unlimited\Folsom\Mixing\Mixing Line 1'
system.mes.notifyEquipmentDataChanged(eqPath)
```

system.mes.object

system.mes.object.filter.createFilter()

Description

Returns a new instance of a MESObjectFilter object for that properties can be set on. This is typically used when a script function requires a MESObjectFilter object as a parameter.

Syntax

system.mes.object.filter.createFilter()

- Parameters

None

- Returns

MESObjectFilter - A new instance of a MESObjectFilter object.

- Scope

All

Code Examples

Code Snippet

```
#Create a filter.
filter = system.mes.object.filter.createFilter()

filter.setEnableStateName('ENABLED')
filter.setMESObjectTypeName('EquipmentClass')
filter.setMESObjectNamePattern('Unload *')
results = system.mes.loadMESObjects(filter)
for link in results:
    print link.getName()
```

Output

```
Unload Station 1
Unload Station 2
```

system.mes.object.filter.parseCustomPropertyValueFilter(filter)

Description

The results can be limited to only include items that have a parse custom property expressions defined by this property that evaluates to true.

Syntax

system.mes.object.filter.parseCustomPropertyValueFilter(filter)

- Parameters

String filter - The property value to filter the results.

- Returns

List<MESPropertyValueFilter> - A list containing property path, type, value, etc.

- Scope

All

Code Examples

Code Snippet

```
filter = system.mes.object.filter.createFilter()
list = system.mes.object.filter.parseCustomPropertyValueFilter(
'pH > 5.0,Width = 2.5')
filter.setCustomPropertyValueFilter(list)
results = system.mes.loadMESObjects(filter)
for link in results:
    print link.getName()
```

system.mes.object.link.create

system.mes.object.link.create(mesObject)

Description

Returns a new instance of a MESObjectLink object for the supplied mesObject. This is typically used when a script function requires a MESObjectLink object as a parameter and the full MES object exists.

Syntax

system.mes.object.link.create(mesObject)

- Parameters

AbstractMESObject mesObject - An MES object object to based the new MESObjectLink. The MES object can be any of the MES objects such as MaterialClass, MaterialLot, OperationSegment, etc. that inherit from AbstractMESObject.

- Returns

A new instance of a MES Object Link object.

Code Examples

Code Snippet

```
obj = system.mes.createMESObject('MaterialClass')
objLink = system.mes.object.link.create(obj)
```

system.mes.object.link.create(mesObjectType, mesObjectUUID)

Description

Returns a new instance of a MES Object Link object for the supplied mesObjectType and mesObjectUUID. This is typically used when a script function requires a MESObjectLink object as a parameter and the MES object type and UUID are known.

Syntax

system.mes.object.link.create(mesObjectType, mesObjectUUID)

- Parameters

MESObjectTypes mesObjectType - The type of MES object to create a link for. See MESObjectTypes for the available types.

String mesObjectUUID - The UUID of MES object to create a link for. See UUIDs for more information.

- Returns

A new instance of a MES Object Link object.

Code Examples

Code Snippet

```
objLink = system.mes.object.link.create('OperationsDefinition',
'a8a3771b-6330-423b-89fb-b54b7933fe3e')
print objLink.getMESObject()
```

Output

```
OperationsDefinition (a8a3771b-6330-423b-89fb-b54b7933fe3e,
Receive Steel, 0 parents, 2 children, 5 custom properties, 2
complex properties)
```

system.mes.object.link.create(mesObjectTypeName, mesObjectUUID)

Description

Returns a new instance of a MESObjectLink object for the supplied mesObjectType and mesObjectUUID. This is typically used when a script function requires a MESObjectLink object as a parameter and the MES object type and UUID are known.

Syntax

system.mes.object.link.create(mesObjectTypeName, mesObjectUUID)

- Parameters

String mesObjectType - The name of the type of MES object to create a link for. See MESObjectTypes for the available types.

String mesObjectUUID - The UUID of MES object to create a link for. See UUIDs for more information.

- Returns

A new instance of a MESObjectLink object.

MES Object Types

MES Objects

Equipment Objects

Material Objects

Personnel Objects

Definition Objects

Request Objects

Response Objects

Code Examples

Code Snippet

```
objLink = system.mes.object.link.create('EquipmentClass', 'd4ba
a7fb-1251-4b9d-8122-a52fc64d4df4')
```

system.mes.object.link.create(mesObjectType, mesObjectUUID, name)

Description

Returns a new instance of a MES Object Link object for the supplied mesObjectType, mesObjectUUID and name. This is typically used when a script function requires a MESObjectLink object as a parameter and the MES object type, UUID and name are known.

Syntax

system.mes.object.link.create(mesObjectType, mesObjectUUID, name)

- Parameters

MESObjectTypes mesObjectType - The type of MES object to create a link for. See MESObjectTypes for the available types.

String mesObjectUUID - The UUID of MES object to create a link for. See UUIDs for more information.

String name - The name of the MES object.

- Returns

A new instance of a MES Object Link object.

Code Examples

Code Snippet

```
objLink = system.mes.object.link.create('StorageUnit', '8da06ff
8-2922-4e0c-a01a-e7cda6899a0e', 'Vinegar Tank 1')
```

system.mes.object.link.create(mesObjectTypeName, mesObjectUUID, name)

Description

Returns a new instance of a MES Object Link object for the supplied mesObjectType name, mesObjectUUID and name. This is typically used when a script function requires a MESObjectLink object as a parameter and the MES object type, UUID and name are known.

Syntax

system.mes.object.link.create(mesObjectTypeName, mesObjectUUID, name)

- Parameters

String mesObjectType - The name of the type of MES object to create a link for. See MESObjectTypes for the available types.

String mesObjectUUID - The UUID of MES object to create a link for. See UUIDs for more information.

String name - The name of the MES object.

- Returns

A new instance of a MES Object Link object.

Code Examples

Code Snippet

```
objLink = system.mes.object.link.create('MaterialDef', 'd4baa7f
b-1251-4b9d-8122-a52fc64d4df4', 'Box')
```

system.mes.object.list.createList

Description

This script function is used to create a list of MES objects.

Syntax

system.mes.object.list.createList()

- Parameters

None

- Returns

The list of MES objects.

- Scope

All

Code Examples

Code Snippet

```
#This code snippet will create a list
m1 = system.mes.loadMESObject('Butterball Turkey', 'MaterialDef
')
m2 = system.mes.loadMESObject('Free Range Turkey', 'MaterialDef
')
objList = system.mes.object.list.createList()
objList.add(m1)
objList.add(m2)
system.mes.saveMESObjects(objList)
```

system.mes.object.parameters.create

Description

This script function creates an instance of MESObjectEventParameters object. See MES Object Event Parameters for more information.

Syntax

system.mes.object.parameters.create()

- Parameters

MES Object Event Parameters - A new instance of a MESObjectEventParameters object.

- Returns

Returns a new instance of a MESObjectEventParameters object that name value pairs can be add to.

Code Snippets

```
mesObject = system.mes.loadMESObject('VIN 3344', 'MaterialLot')

if mesObject != None:
    params = system.mes.object.parameters.create()
#Add parameters for the new instance.
    params.put('Kind', 'Dressing')
    params.put('Priority', 'High')
    system.mes.executeMESEvent(mesObject, 'My User Event', params)
```

system.mes.pasteSchedule

Description

Paste an operations schedule and all associated operations requests and request segments objects.

Syntax

system.mes.pasteSchedule(mesObjectList, equipmentLink, preferredStart)

- Parameters

MESObjectList mesObjectList - The list containing the operations schedule and associated operations requests and request segments objects to paste.

MESObjectLink equipmentLink - A link representing MES equipment object to schedule the first operations request for.

Date preferredStart - The date of the preferred start time to schedule the first operations request for.

- Returns

The list containing the operations schedule and associated operations requests and request segments objects that have been scheduled. The list is returned as a MES Object List object that is a collection holding MES objects.

- Scope

All

Code Examples

Code Snippet

```
from java.util import Calendar

begin = Calendar.getInstance()
begin.add(Calendar.DAY_OF_MONTH, -30)
start = begin.getTime()

#Get the MES object link of Unload Station 1
eqPath = 'My Enterprise\California\Receiving\Unload Station 1'
eqLink = system.mes.getMESObjectLinkByEquipmentPath(eqPath)

#Get the MES object list containing the schedule details
schedule = system.mes.loadSchedule('14850859-fe9e-4aa8-a9d2-
856022ef1bb3')
system.mes.pasteSchedule(schedule, eqLink, start)
```

system.mes.property.namevalue.createInstance

Description

Returns an MES property list.

Syntax

system.mes.property.namevalue.createInstance()

- Parameters

None

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet
```

system.mes.property.validateValueString

Description

Validates the candidate value against the type definition.

Syntax

system.mes.property.validateValueString(val, type)

- Parameters

String val - The string to be validated.

DataType type - The type to check the validation for.

- Returns

Boolean True when the value is valid for the type, otherwise false.

- Scope

All

Code Examples

```
Code Snippet
```

system.mes.removeTagCollectorValue

Description

Remove the value with the specified timestamp for the MES tag collector. If a value does not existing for the specified timestamp, then an exception will be returned.

Syntax

system.mes.removeTagCollectorValue(equipmentPath, collectorType, key, dateTime)

- Parameters

String equipmentPath - The path from the production model to the desired equipment.

String collectorType - The name of the tag collector type. See Tag Collector Types for more details.

String key - Where there are multiple i nstances of a tag collector type, this specifies which one to use . For example, there can be multiple MES counters for the "Equipment Count" tag collector type. If not needed, pass an empty string. In other words it is the name of the MES counter or the name of the Additional Factor.

Date dateTime - The date of the value to remove.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet

date = system.date.getDate(2017, 2, 20)
dateTime = system.date.setTime(date, 15, 13, 31)
equipmentPath = '[global]\Enterprise\San Marcos\MP Rotator\MP
Rotator 1'
collectorType = 'Equipment Mode'
key = ''

system.mes.removeTagCollectorValue(equipmentPath,
collectorType, key, dateTime)
```

Overview

These script functions are used to remove all values within the specified range or from a list of timestamps for the MES tag collector.

Method Options

system.mes.removeTagCollectorValues(equipmentPath, collectorType, key, dateTimeList)

Description

Remove all values within the specified range or from a list of timestamps for the MES tag collector. When a list of timestamps are provided and one of the timestamps does not exist, then an exception will be returned.

Syntax

system.mes.removeTagCollectorValues(equipmentPath, collectorType, key, dateTimeList)

- Parameters

String equipmentPath - The path from the production model to the desired equipment.

String collectorType - The name of the tag collector type. See Tag Collector Types for more details.

String key - Where there are multiple i nstances of a tag collector type, this specifies which one to use . For example, there can be multiple MES counters for the "Equipment Count" tag collector type. If not needed, pass an empty string. In other words it is the name of the MES counter or the name of the Additional Factor.

PyList dateTimeList - A Python list containing dates(of type Date) to remove from the MES tag collector.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
date = system.date.getDate(2017, 2, 20)
dateTime1=system.date.setTime(date, 14, 40, 59)
dateTime2=system.date.setTime(date, 14, 43, 46)
dateTime3=system.date.setTime(date, 14, 54, 11)
equipmentPath = '[global]\Enterprise\San Marcos\MP Rotator\MP
Rotator 1'
collectorType = 'Equipment Mode'
key = ''
dateTimeList = [dateTime1, dateTime2, dateTime3]
system.mes.removeTagCollectorValues(equipmentPath,
collectorType, key, dateTimeList)
```

system.mes.removeTagCollectorValues(equipmentPath, collectorType, key, beginDateTime, endDateTime)

Description

Remove all values within the specified range or from a list of timestamps for the MES tag collector. When a list of timestamps are provided and one of the timestamps does not exist, then an exception will be returned.

Syntax

system.mes.removeTagCollectorValues(equipmentPath, collectorType, key, beginDateTime, endDateTime)

- Parameters

String equipmentPath - The path from the production model to the desired equipment.

String collectorType - The name of the tag collector type. See Tag Collector Types for more details.

String key - Where there are multiple i nstances of a tag collector type, this specifies which one to use . For example, there can be multiple MES counters for the "Equipment Count" tag collector type. If not needed, pass an empty string. In other words it is the name of the MES counter or the name of the Additional Factor.

Date beginDateTime - The starting date of values that will be removed.

Date endDateTime - The ending date of values that will be removed.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Set the start and end time
date = system.date.getDate(2017, 2, 21)
dateTime1 = system.date.setTime(date, 14, 45, 15)
dateTime2 = system.date.setTime(date, 14, 49, 15)
equipmentPath = '[global]\Enterprise\San Marcos\MP Rotator\MP
Rotator 1'
collectorType = 'Equipment State'
key = ''
system.mes.removeTagCollectorValues(equipmentPath,
collectorType, key, dateTime1, dateTime2)
```

system.mes.resetScheduleStatus

Description

Reset the state of the operation request, based on the operationsRequestUUID parameter, to AUTO_PENDING. If an error occurred during automatic start of a operation request, this script function allows it to be reset so that another automatic attempt can be done.

Syntax

system.mes.resetScheduleStatus(operationsRequestUUID)

- Parameters

String operationsRequestUUID - The UUID of the operations request to reset.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#The following is a code snippet to reset the status of a
schedule.
system.mes.resetScheduleStatus('5546bd58-ea2a-4acd-b9d3-
feb5de7e2085')
```

system.mes.saveMESObject

Description

Save the MES object passed mesObject parameter. This will update the active object in memory of the Ignition server and save the settings to the database.

Syntax

system.mes.saveMESObject(mesObject)

- Parameters .

AbstractMESObject mesObject - The MES object to save. See AbstractMESObject object in the MES documentation.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#To save MES object.
matClass = system.mes.createMESObject('MaterialClass')
matClass.setPropertyValue('Name', 'Turkey')
matClass.addCustomProperty('Weight', 'Float8', 'Weight of the
turkey', 'Lbs', True, True)
system.mes.saveMESObject(matClass)
```

system.mes.saveMESObjects

Description

Save one or more MES objects. The MESObjectList parameter object holds a collections of MES objects to save. This will update the active objects in cache on the Ignition server and save the settings to the database.

Syntax

system.mes.saveMESObjects(mesObjectList)

- Parameters

MES Object List mesObjectList - A list that holds the collection of MES objects to save.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#This code will save the list of objects
filter = system.mes.object.filter.createFilter()
filter.setMESObjectNamePattern('* Turkey')
objList = system.mes.loadMESObjects(filter)
system.mes.saveMESObjects(objList)
```

system.mes.saveSchedule

Description

Save the operations schedule object and all other associated MES objects contained in the list passed in the mesObjectList parameter. This list generally includes the Operations Schedule and (a) Request Segment(s) if preceeded by the system.mes.createSchedule() function.

Syntax

system.mes.saveSchedule(mesObjectList)

- Parameters

MES Object List mesObjectList - The list containing all schedule related MES objects to save.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#This code will save the list containing schedules of MES
objects
filter = system.mes.object.filter.createFilter()
filter.setMESObjectTypeName('OperationsSchedule')
scheduleList = system.mes.loadMESObjects(filter)
system.mes.saveSchedule(scheduleList)
```

MES Object List

Object Description

The **MESObjectList** is a collection of MES objects. A list may contain any number of mes objects. From the MESObjectList object an MES object with a specific uuid can be loaded by calling the findByUUID() function.

Scripting Functions

The following function can be used to create an **MESObjectList** object.

system.mes.object.list.createList()

Description

This script function is used to create a list of MES objects.

Syntax

system.mes.object.list.createList()

- Parameters

None

- Returns

The list of MES objects.

- Scope

All

Example

```
obj = system.mes.loadMESObject('Box', 'MaterialDef')
objList = system.mes.object.list.createList()
objList.add(obj)
```

Methods

The following methods exist for the MES Object List.

add(mesobject)

Description

This script function is used to add a MES object to the list.

Syntax

add(mesobject)

- Parameters

AbstractMESObject mesobject - The object to be added.

- Returns

True if the object is added and False otherwise.

- Scope

All

remove(mesobject)

Description

This script function is used to remove a MES object to the list.

Syntax

remove(mesobject)

- Parameters

AbstractMESObject mesobject - The object to be removed.

- Returns

True if the object is removed and False otherwise.

- Scope

All

addAll(collection)

Description

Appends all of the elements in the specified collection to the end of this MES object list.

Syntax

addAll(collection)

- Parameters

List collection - A collection containing elements to be added to this list.

- Returns

True if all the objects are added to this MES object list and False otherwise.

- Scope

All

removeAll(collection)

Description

Removes all of the elements from the MES object list.

Syntax

removeAll(collection)

- Parameters

List collection - A collection containing elements to be removed to this list.

- Returns

True if the objects in the list is removed and False otherwise.

- Scope

All

findByUUID()

Description

Find a specific object from a list of MES objects by UUID.

Syntax

findByUUID(uuid)

- Parameters

String uuid - UUID of the MES object.

- Returns

AbstractMESObject mesObject - The MES object corresponding to the specific UUID.

- Scope

All

hasSingleMESObject()

Description

Checks whether the list contains more than one MES object.

Syntax

hasSingleMESObject()

- Parameters

None

- Returns

Boolean

- Scope

All

Code Example

```
Code Snippet
```

```
#Creates a list of MES objects
objList = system.mes.object.list.createList()

#Load the objects to be added
m1 = system.mes.loadMESObject('Bulk Almonds', 'MaterialDef')
m2 = system.mes.loadMESObject('Bulk Peanuts', 'MaterialDef')

#Adds the objects to list
objList.add(m1)
objList.add(m2)

#Save the changes
system.mes.saveMESObjects(objList)

#This code snippet will check if the list contains only one
MES object
print objList.hasSingleMESObject()

#Gets info about the object specified by the uuid
print objList.findByUUID('a3f05165-1cee-4661-a1e8-
d282bf2c6a02')

#Creates a filter
filter = system.mes.object.filter.createFilter()
filter.setEnableStateName('ENABLED')
filter.setMESObjectNamePattern('Receive *')
mesList = system.mes.loadMESObjects(filter)

#Adds the elements in list 'mesList' to the list 'objList'
objList.addAll(mesList)

#Removes all the objects from the list
objList.removeAll(mesList)
```

Output

```
True
True
False
MaterialDef (a3f05165-1cee-4661-a1e8-d282bf2c6a02, Bulk
Almonds, 1 parents, 0 children, 2 custom properties, 0 compl
ex properties)
True
True
```

Overview

These script functions are used to schedule the operations.

The two functions represent different scheduling needs.

- For fixed duration (example: schedule an operation for an entire shift), the first signature which defines a begin and end time is appropriate.
- To utilize the scheduling engine's capability for estimating (example: incorporating scheduled down time for the line, etc.), the second signature which defines just a preferred start time is appropriate.

Method Options

system.mes.scheduleOperations(mesObjectList, begin, end, allowOverlapping, category)

Description

Schedule the operations schedule and associated operations request and request segment objects contained in the mesObjectList parameter. The scheduled duration of each request segment will be based on the target quantity.

Note that the beginning and end time are set in this function call and will override schedule rates.

Syntax

system.mes.scheduleOperations(mesObjectList, begin, end, allowOverlapping, category)

- Parameters

MESObjectList mesObjectList - A list containing the operations schedule and associated operations request and request segment objects to schedule.

Date begin - The date to schedule the first operations request for.

Date end - The date to complete the last operations request at.

Boolean allowOverlapping - If True, allow schedule entries to overlap if needed.

String category - The category to use when scheduling.

- Returns

The list containing the operations schedule and associated operations requests and request segments objects that have been scheduled. The list is returned as a MES Object List object that is a collection holding MES objects.

- Scope

All

Code Examples

Code Snippet

```
#This code will print the list of operations that are
scheduled.
from java.util import Calendar
beginCal = Calendar.getInstance()
beginCal.add(Calendar.DAY_OF_MONTH, -30)
begin = beginCal.getTime()
endCal = Calendar.getInstance()
endCal.add(Calendar.DAY_OF_MONTH, -1)
end = endCal.getTime()
objList = system.mes.loadSchedule('14850859-fe9e-4aa8-a9d2-
856022ef1bb3')
scheduleList = system.mes.scheduleOperations(objList, begin,
end, True, 'Actual')
for ndx in range(scheduleList.size()):
    scheduleOperations = scheduleList.get(ndx)
    print scheduleOperations
```

Output

```
OperationsSchedule (14850859-fe9e-4aa8-a9d2-856022ef1bb3,
Receive Steel (1) Schedule, 0 parents, 0 children, 0 custom
properties, 1 complex properties)
OperationsRequest (9ce541f3-6b9a-47f2-8ca5-59f0bdad81e8,
Receive Steel (1), 0 parents, 0 children, 0 custom properties,
2 complex properties)
RequestSegment (b8c416b2-17a3-4f04-8842-b24854511d89, Receive
Steel (1), 0 parents, 0 children, 0 custom properties, 6 comple
x properties)
```

system.mes.scheduleOperations(mesObjectList, preferredStart, allowOverlapping, category)

Description

Schedule the operations schedule and associated operations request and request segment objects contained in the mesObjectList parameter. The scheduled duration of each request segment will be based on the target quantity and the configured segment's schedule rate.

Syntax

system.mes.scheduleOperations(mesObjectList, preferredStart, allowOverlapping, category)

- Parameters

MESObjectList mesObjectList - A list containing the operations schedule and associated operations request and request segment objects to schedule.

Date preferredStart - The date of the preferred start time to schedule the first operations request for.

Boolean allowOverlapping - If True, allow schedule entries to overlap if needed.

String category - The category to use when scheduling.

- Returns

The list containing the operations schedule and associated operations requests and request segments objects that have been scheduled. The list is returned as a MES Object List object that is a collection holding MES objects.

- Scope

All

Code Examples

Code Snippet

```
#This code will print the list of operations that are
scheduled.
from java.util import Calendar
beginCal = Calendar.getInstance()
beginCal.add(Calendar.DAY_OF_MONTH, -30)
preferredStart = beginCal.getTime()
```

```
objList = system.mes.loadSchedule('14850859-fe9e-4aa8-a9d2-
856022ef1bb3')
scheduleList = system.mes.scheduleOperations(objList,
preferredStart, True, 'Actual')
for ndx in range(scheduleList.size()):
    scheduleOperations = scheduleList.get(ndx)
    print scheduleOperations
```

Output

```
OperationsSchedule (14850859-fe9e-4aa8-a9d2-856022ef1bb3,
Receive Steel (1) Schedule, 0 parents, 0 children, 0 custom
properties, 1 complex properties)
OperationsRequest (9ce541f3-6b9a-47f2-8ca5-59f0bdad81e8,
Receive Steel (1), 0 parents, 0 children, 0 custom properties,
2 complex properties)
RequestSegment (b8c416b2-17a3-4f04-8842-b24854511d89, Receive
Steel (1), 0 parents, 0 children, 0 custom properties, 6 comple
x properties)
```

system.mes.searchMESObjects

Description

Search for MES objects that meet the criteria of the filter parameter.

Syntax

system.mes.searchMESObjects(filter)

- Parameters

MESObjectFilter filter - A filter containing the criteria to select MES object to return.

- Returns

A list of MES Object Link objects that meet the criteria specified in the filter parameter.

- Scope

All

Code Examples

Code Snippet

```
#This snippet will print only the mesObjects that satisfies
the specific constrain
filter = system.mes.object.filter.createFilter()
filter.setMESObjectNamePattern('Vinegar')
list = system.mes.searchMESObjects(filter)
for ndx in range(list.size()):
    mesObject = list.get(ndx)
    print mesObject.getMESObjectType().getDisplayName()
```

Output

```
Material Class
```

Code Examples

Code Snippet

```
#This snippet will return cell names for
fltr = system.mes.object.filter.createFilter()

typeName = 'LineCell' #typeName can be 'Site', 'Area', 'Line'
, 'LineCell' ....

fltr.setMESObjectTypeName('LineCell')

linePath = ''[global]\Your Enterprise\Site 1\Packaging\Line 1'

fltr.setPrimaryMESObjectPath(linePath)

list = system.mes.searchMESObjects(fltr)
for ndx in range(list.size()):
    print list.get(ndx)
```

Output

```
2
```

system.mes.splitSchedule

Description

Split one or more operations requests off from an existing operations schedule into a new operations schedule. The split point is from the specified operations request, based on the operationsRequestUUID parameter, to the last operations request of the route. Operations schedules that only have one operations request cannot be split.

Syntax

system.mes.splitSchedule(operationsScheduleUUID, operationsRequestUUID)

- Parameters

String operationsScheduleUUID - The UUID of the operations schedule object that will be split.

String operationsRequestUUID - The UUID of one of the operations request within the operations schedule to split at.

- Returns

The UUID of the new operations schedule.

- Scope

All

Code Examples

Code Snippet

```
#The following is a snippet to split the schedule for the
requested operation.
system.mes.splitSchedule('928fa0e2-f739-427b-bd88-464062396d8a'
, '1243612e-71fd-4fc4-82a3-18c4ccd71571')
```

system.mes.synchronizeMESPersonnel

Description

Read users from the Ignition user source profile and synchronize with the MESPerson objects. This happens automatically on an hourly basis.

Syntax

system.mes.synchronizeMESPersonnel( )

- Parameters

None

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#This code read the users from source profile and synchronize
it with MESPerson objects
sync = system.mes.synchronizeMESPersonnel()
```

system.mes.updateDependencies

Description

Update the changes of the specified MES object to the MES object represented by the MESObjectLinks in the mesObjectLinkList parameter.

Syntax

system.mes.updateDependencies(mesObject, mesObjectLinkList)

- Parameters

AbstractMESObject mesObject - A MES object with modified properties are propagated to the dependent MES object. See AbstractMESObject object in the MES documentation.

MESList mesObjectLinkList - A list containing MES object links that represent the MES objects to propagate the changes to. This list can be obtained by calling getDependencies. Only the MES object links in this list that have not been disabled will be updated. See MES Object Link in the MES documentation.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
obj = system.mes.loadMESObject('Receive Turkey', 'ProcessSegmen
t')
objList = system.mes.getDependencies(obj)
system.mes.updateDependencies(obj, objList)
```

system.mes.updateSegment

Description

This method is used to update information for a actively running segment. Information that maybe updated is limited to properties that are appropriate to be changed. For example, equipment cannot be changed because a new operation and segment must be created for them. However, lots can be changed. For example, if a raw material lot runs out, then and new lot can be used to complete out the production run.

Syntax

system.mes.updateSegment(responseSegment)

- Parameters

MESResponseSegment responseSegment - The MES object to update.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
defSeg = system.mes.loadMESObject('Receive Material', 'Operatio
nsSegment')
eqPath = 'My Enterprise\California\Receiving\Unload Station 1'
respSeg = system.mes.createSegment(defSeg, eqPath)
seg.setMaterial('In Steel Type', 1000.0)
system.mes.updateSegment(respSeg)
```

system.mes.updateTagCollectorLastValue

Description

Update the last chronological value recorded by the MES tag collector.

Syntax

system.mes.updateTagCollectorLastValue(equipmentPath, collectorType, key, value)

- Parameters

String equipmentPath - The path from the production model to the desired equipment.

String collectorType - The name of the tag collector type. See Tag Collector Types for more details.

String key - Where there are multiple i nstances of a tag collector type, this specifies which one to use . For example, there can be multiple MES counters for the "Equipment Count" tag collector type. If not needed, pass an empty string. In other words it is the name of the MES counter or the name of the Additional Factor.

Datatype value - The last value recorded for the MES tag collector will be updated with this value.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Set the start and end time
date = system.date.getDate(2017, 2, 21)
beginDateTime = system.date.setTime(date, 15, 54, 59)
endDateTime = system.date.now()

equipmentPath = '[global]\Enterprise\San Marcos\MP Rotator\MP
Rotator 1'
collectorType = 'Equipment State'
key = ''
system.mes.updateTagCollectorLastValue(equipmentPath,
collectorType, key, 2)
#get the tag values
data= system.mes.getTagCollectorValues(equipmentPath,
collectorType, key, beginDateTime, endDateTime)
for row in range(data.rowCount):
   for col in range(data.columnCount):
      print data.getValueAt(row, col)
```

Output

```
Tue Mar 21 15:55:15 PDT 2017
4
3
None
None
Tue Mar 21 15:55:15 PDT 2017
4
3
None
None
Tue Mar 21 15:55:15 PDT 2017
4
3
None
None
Tue Mar 21 15:58:15 PDT 2017
2
3
None
None
Tue Mar 21 15:58:15 PDT 2017
2
5
None
None
Tue Mar 21 15:58:15 PDT 2017
2
5
None
None
Tue Mar 21 15:58:15 PDT 2017
2
5
None
None
```

Overview

These script functions are used to update the value specified by the date and time for the MES tag collector.

The Equipment State tag collector has the following auxiliary values.

- EquipmentUUID - The unique identifier for the equipment.
- State - The current equipment state.
- OriginalState - The original equipment state before its updation.

- DifferedToUUID - If the original EquipmentUUID is changed using the Downtime Table then the new uuid is DifferedToUUID.
- DifferedState - If the original state is changed using the Downtime Table then the new state is DifferedState.

Method Options

system.mes.updateTagCollectorValue(equipmentPath, collectorType, key, dateTime, value)

Description

Update the value specified by the date and time for the MES tag collector. If a value does not existing for the specified timestamp, then an exception will be returned.

Syntax

system.mes.updateTagCollectorValue(equipmentPath, collectorType, key, dateTime, value)

- Parameters

String equipmentPath - The path from the production model to the desired equipment.

String collectorType - The name of the tag collector type. See Tag Collector Types for more details.

String key - Where there are multiple i nstances of a tag collector type, this specifies which one to use . For example, there can be multiple MES counters for the "Equipment Count" tag collector type. If not needed, pass an empty string. In other words it is the name of the MES counter or the name of the Additional Factor.

Date dateTime - The date of the value to update.

Datatype value - The MES tag collector will be updated with this value.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Set the start and end time
date = system.date.getDate(2017, 2, 21)
beginDateTime = system.date.setTime(date, 15, 58, 15)
endDateTime = system.date.now()
equipmentPath = '[global]\Enterprise\San Marcos\MP Rotator\MP
Rotator 1'
collectorType = 'Equipment State'
key = ''
#update the tag values
system.mes.updateTagCollectorValue(equipmentPath,
collectorType, key, beginDateTime, 9)
#get the tag values
data= system.mes.getTagCollectorValues(equipmentPath,
collectorType, key, beginDateTime, endDateTime)
for row in range(data.rowCount):
   for col in range(data.columnCount):
      print data.getValueAt(row, col)
```

Output

```
Tue Mar 21 15:58:15 PDT 2017
9
8
None
None
Tue Mar 21 15:58:15 PDT 2017
9
5
None
None
Tue Mar 21 15:58:15 PDT 2017
9
5
None
None
Tue Mar 21 15:58:15 PDT 2017
9
5
None
None
```

system.mes.updateTagCollectorValue(equipmentPath, collectorType, auxValueName, key, dateTime, value)

Description

Update the value specified by the date and time for the MES tag collector. If a value does not existing for the specified timestamp, then an exception will be returned.

Syntax

system.mes.updateTagCollectorValue(equipmentPath, collectorType, auxValueName, key, dateTime, value)

- Parameters

String equipmentPath - The path from the production model to the desired equipment.

String collectorType - The name of the tag collector type. See Tag Collector Types for more details.

String auxValueName - This specifies which auxiliary value to update. For example, the Equipment State tag collector has an "OriginalState" auxiliary value. See Auxiliary value for more information.

String key - Where there are multiple i nstances of a tag collector type, this specifies which one to use . For example, there can be multiple MES counters for the "Equipment Count" tag collector type. If not needed, pass an empty string. In other words it is the name of the MES counter or the name of the Additional Factor.

Date dateTime - The date of the value to update.

Datatype value - The MES tag collector will be updated with this value.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#Set the start and end time
date = system.date.getDate(2017, 2, 21)
dateTime = system.date.setTime(date, 15, 58, 15)
endDateTime = system.date.now()
equipmentPath = '[global]\Enterprise\San Marcos\MP Rotator\MP
Rotator 1'
collectorType = 'Equipment State'
key = ''
system.mes.updateTagCollectorValue(equipmentPath,
collectorType, 'DifferedState', key, dateTime, 4)
#get the tag values
data= system.mes.getTagCollectorValues(equipmentPath,
collectorType, key, dateTime, endDateTime)
for row in range(data.rowCount):
   for col in range(data.columnCount):
      print data.getValueAt(row, col)
```

Output

```
Tue Mar 21 15:58:15 PDT 2017
9
2
None
4
Tue Mar 21 15:58:15 PDT 2017
9
5
None
4
Tue Mar 21 15:58:15 PDT 2017
9
5
None
4
Tue Mar 21 15:58:15 PDT 2017
9
5
None
4
```

system.mes.updateTagCollectorValues

Description

Update multiple values recorded by the MES tag collector. If a value does not existing for one of the timestamps, then an exception will be returned.

Syntax

system.mes.updateTagCollectorValues(equipmentPath, collectorType, key, values)

- Parameters

String equipmentPath - The path from the production model to the desired equipment.

String collectorType - The name of the tag collector type. See Tag Collector Types for more details.

String key - Where there are multiple i nstances of a tag collector type, this specifies which one to use . For example, there can be multiple MES counters for the "Equipment Count" tag collector type. If not needed, pass an empty string. In other words it is the name of the MES counter or the name of the Additional Factor.

PyDictionary values - A Python dictionary containing the date (of type Date) and value (Refer Datatype) pairs to update.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
date1 = system.date.getDate(2017, 2, 17)
dateTime1 = system.date.setTime(date1, 12, 55, 31)
date2 = system.date.getDate(2017, 2, 17)
```

```
dateTime2 = system.date.setTime(date2, 13, 16, 13)

equipmentPath = '[global]\Enterprise\San Marcos\MP
Rotator\Test Line 1'
collectorType = 'Equipment Mode'
key = ''
values={dateTime1:2, dateTime2:2}
system.mes.updateTagCollectorValues(equipmentPath,
collectorType, key, values)
```

system.mes.validateMESObjectName

Description

Validates the proposed name for the specified MES object type. If the name is not valid, an exception will be thrown.

Syntax

system.mes.validateMESObjectName(mesObjectTypeName,mesObjectUUID,name)

- Parameters

String MES Object Type Name - The MES object type to validate the name for. Because names must be unique amount the category of MES object type, this is required. See MES Object Type Name for more details.

String mesObjectUUID - When validating the name of an existing MES object, this is the UUID for it so that it will ignore duplicate name check with itself. Otherwise, just pass an empty string.

String name - Proposed name to validate

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#This is an example for the validation.
 system.mes.validateMESObjectName('StorageZone','ccd29f94-953a-4343-a9da-912755f60abe','Vinegar Tank 1')
```

9.7.3 system.mes.analysis

Available system.mes.analysis functions

system.mes.analysis.createMESAnalysisSettings

Description

Create a new analysis setting object.

Syntax

system.mes.analysis.createMESAnalysisSettings(savedSettingsName)

- Parameters

String savedSettingsName - The name to the new analysis settings.

- Returns

MESAnalysisSettings - A new MESAnalysisSettings object used to store data points, filter expressions, group by, order by, etc.

- Scope

All

Code Examples

Code Snippet

```
##Note that the Analysis Settings objects should be uniquely
named.
##Therefore, it's best practice to check first before
attempting to create a new one.
sasName = 'SAS Test'
list = system.mes.analysis.getMESAnalysisSettingsList()
if sasName not in list:
    sasObj = system.mes.analysis.createMESAnalysisSettings
(sasName)
    print sasObj
    system.mes.saveMESObject(sasObj)
else:
    print 'Analysis name already in use!  Pick unique name.'
```

Output

```
 AnalysisSettings (6bd5b603-ab27-4a13-a01d-f8b581de4409, SAS
Test, 0 parents, 0 children, 0 custom properties, 2 complex
properties)
```

system.mes.analysis.deleteMESAnalysisSettings

Description

Deletes the specified stored analysis settings.

Syntax

system.mes.analysis.deleteMESAnalysisSettings(savedSettingsName)

- Parameters

String savedSettingsName - The name of the saved analysis settings to delete.

- Returns

None

- Scope

All

Code Examples

Code Snippet

```
sasName = 'SAS Test'
list = system.mes.analysis.getMESAnalysisSettingsList()
print list

system.mes.analysis.deleteMESAnalysisSettings('SAS Test')
list = system.mes.analysis.getMESAnalysisSettingsList()
print list
```

Output

```
[SAS Test]
[]
```

Overview

These script functions are used to execute the analysis specified in the parameters.

Method Options

system.mes.analysis.executeAnalysis(beginDate, endDate, settings)

Description

Execute and returns the results for the analysis specified in the parameters.

Syntax

system.mes.analysis.executeAnalysis(beginDate, endDate, settings)

- Parameters

Date beginDate - Date object containing the beginning date to based the analysis results on.

Date endDate - Date object containing the ending date to based the analysis results on.

MESAnalysisSettings settings - MESAnalysisSettings object containing the data points, filter expressions, group by, order by, etc. setting to return the results for.

- Returns

MESAnalysisResults - A MESAnalysisResults object containing the results of the analysis that was returned. (See MESAnalysisResults in the manual for more details.)

- Scope

All

Code Examples

Code Snippet

```
##Execute the "Downtime" Analysis Settings and print the
dataset object.
sasName = 'Downtime'
obj = system.mes.analysis.getMESAnalysisSettings(sasName)
end = system.date.now()
start = system.date.addDays(end, -2)
result = system.mes.analysis.executeAnalysis(start, end, obj)
print result.getDataset()
```

Output

```
Dataset [5R x 12C]
```

system.mes.analysis.executeAnalysis(beginDate, endDate, settings, parameters)

Description

Execute and returns the results for the analysis specified in the parameters.

Syntax

system.mes.analysis.executeAnalysis(beginDate, endDate, settings, parameters)

- Parameters

Date beginDate - Date object containing the beginning date to based the analysis results on.

Date endDate - Date object containing the ending date to based the analysis results on.

MESAnalysisSettings settings - MESAnalysisSettings object containing the data points, filter expressions, group by, order by, etc. setting to return the results for.

PyDictionary parameters - A PyDictionary containing name / value pairs for each parameter that exists in the analysis settings.

- Returns

MESAnalysisResults - A MESAnalysisResults object containing the results of the analysis that was returned. (See MESAnalysisResults in the manual for more details.)

- Scope

All

Code Examples

Code Snippet

```
##Execute the "Downtime" Analysis Settings and print the
dataset object.
sasName = 'Downtime'
obj = system.mes.analysis.getMESAnalysisSettings(sasName)
end = system.date.now()
start = system.date.addDays(end, -2)
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
params = {'EqPath' : eqPath, 'PackageCount' : 1.0}
result = system.mes.analysis.executeAnalysis(start, end, obj,
params)
print result.getDataset()
```

Output

```
Dataset [3R x 12C]
```

system.mes.analysis.executeAnalysis(beginDate, endDate, savedSettingsName)

Description

Execute and returns the results for the analysis specified in the parameters.

Syntax

system.mes.analysis.executeAnalysis(beginDate, endDate, savedSettingsName)

- Parameters

Datatype - Date object containing the beginning date to based the analysis results on.

Datatype - Date object containing the ending date to based the analysis results on.

String savedSettingsName - The name of the saved analysis settings to execute and return the results for.

- Returns

MESAnalysisResults - A MESAnalysisResults object containing the results of the analysis that was returned. (See MESAnalysisResults in the manual for more details.)

- Scope

All

Code Examples

Code Snippet

```
##Execute the "Downtime" Analysis Settings and print the
dataset object.
sasName = 'Downtime'
obj = system.mes.analysis.getMESAnalysisSettings(sasName)
end = system.date.now()
start = system.date.addDays(end, -2)
result = system.mes.analysis.executeAnalysis(start, end,
sasName)
print result.getDataset()
```

```
Output

Dataset [5R x 12C]
```

system.mes.analysis.executeAnalysis(beginDate, endDate, savedSettingsName, parameters)

Description

Execute and returns the results for the analysis specified in the parameters.

Syntax

system.mes.analysis.executeAnalysis(beginDate, endDate, savedSettingsName, parameters)

- Parameters

Datatype - Date object containing the beginning date to based the analysis results on.

Datatype - Date object containing the ending date to based the analysis results on.

String savedSettingsName - The name of the saved analysis settings to execute and return the results for.

PyDictionary parameters - A PyDictionary containing name / value pairs for each parameter that exists in the analysis settings.

- Returns

MESAnalysisResults - A MESAnalysisResults object containing the results of the analysis that was returned. (See MESAnalysisResults in the manual for more details.)

- Scope

All

Code Examples

Code Snippet

```
##Execute the "Downtime" Analysis Settings and print the
dataset object.
sasName = 'Downtime'
obj = system.mes.analysis.getMESAnalysisSettings(sasName)
end = system.date.now()
start = system.date.addDays(end, -2)
eqPath = '[global]\Dressings Inc\California\Raw
Materials\Unload Station 1'
params = {'EqPath' : eqPath, 'PackageCount' : 1.0}
result = system.mes.analysis.executeAnalysis(start, end,
sasName, params)
print result.getDataset()
```

Output

```
Dataset [3R x 12C]
```

system.mes.analysis.getDataPointOptions

Description

Return data point options that can be used when executing analysis.

Syntax

system.mes.analysis.getDataPointOptions(groupFilter, itemFilter)

- Parameters

String groupFilter - A filter to limit the data point options returned to one or more groups. Multiple groups can be specified by separating them with commas. The wildcard * is accepted.

String itemFilter - A filter to limit the data point options returned to one or more items. Multiple data point items can be specified by separating them with commas. The wildcard * is accepted.

- Returns

List<AbstractValueItemInfo> - Returns a map (a key-value pair) containing the filter group path as the key and a list of AbstractValueItemInfo objects as the value. See AbstractValueItemInfo object documentation for details.

- Scope

All

Code Examples

Code Snippet

```
##Get a list of data point options for the Equipment group:
list = system.mes.analysis.getDataPointOptions('Equipment', '*'
)
for item in list:
    for x in list[item]:
        print item, '::', x.getName()
```

Output

```
Equipment :: Product Code
Equipment :: Work Order
Equipment :: Is Key Cell
Equipment :: Equipment Type
Equipment :: Equipment Name
Equipment :: Operation UUID
Equipment :: Equipment Path
Equipment :: Equipment Cell Order
Equipment :: Rate Period
```

system.mes.analysis.getFilterOptions

Description

Return filter options that can be used when executing analysis.

Syntax

system.mes.analysis.getFilterOptions(groupFilter, itemFilter)

- Parameters

String groupFilter - A filter to limit the filter options returned to one or more groups. Multiple groups can be specified by separating them with commas. The wildcard * is accepted.

String itemFilter - A filter to limit the group by options returned to one or more items. Multiple filter items can be specified by separating them with commas. The wildcard * is accepted.

- Returns

List <AbstractValueItemInfo> - Returns a map (a key-value pair) containing the filter group path as the key and a list of AbstractValueItemInfo objects as the value. See AbstractValueItemInfo object documentation for details.

- Scope

All

Code Examples

Code Snippet

```
##Get a list of filter options from the OEE group:
list = system.mes.analysis.getFilterOptions('OEE', '*')
for item in list:
    for x in list[item]:
        print item, '::', x.getName()
```

Output

```
OEE :: OEE Infeed Count Equipment Path
OEE :: Target Changeover Time
OEE :: OEE
OEE :: Standard Rate
OEE :: Elapsed Time
OEE :: OEE Outfeed Count Equipment Path
OEE :: Schedule Rate
```

system.mes.analysis.getFilterValues

Description

Return values that a filter item can be set to when executing analysis. For filter items such as a date, no filter values will be returned.

Syntax

system.mes.analysis.getFilterValues(filterName, beginDate, endDate)

- Parameters

String filterName - The name of the filter item to return values for.

Date beginDate - The starting date to limit the values to.

Date endDate - The ending date to limit the values to.

- Returns

List<String> - Returns a list of strings. Each string is a possible value that the filter can be set to.

- Scope

All

Code Examples

Code Snippet

```
##Print the Filter Values when filtering by Work Order
end = system.date.now()
start = system.date.addDays(end, -2)
result = system.mes.analysis.getFilterValues('Work Order',
start, end)
print result
```

Output

```
[WO 1 - PC_0001, WO 2 - PC-002, TestWO, TestWO2]
```

system.mes.analysis.getGroupByOptions

Description

Return group-by options that can be used when executing analysis.

Syntax

system.mes.analysis.getGroupByOptions(groupFilter, itemFilter)

- Parameters

String groupFilter - A filter to limit the group-by options returned to one or more groups. Multiple groups can be specified by separating them with commas.

String itemFilter - A filter to limit the group-by options returned to one or more items. Multiple group-by items can be specified by separating them with commas.

- Returns

List<AbstractValueItemInfo> - Returns a map (a key-value pair) containing the filter group path as the key and a list of AbstractValueItemInfo objects as the value. See AbstractValueItemInfo object documentation for details.

- Scope

All

Code Examples

Code Snippet

```
#Gets groupby options
options = system.mes.analysis.getGroupByOptions('*Downtime', 'L
ine *')
for item in options:
    for x in options[item]:
```

```
        print item, '::', x.getName()
```

Output

```
Equipment/Line/Downtime :: Line Downtime Occurrence Count
Equipment/Line/Downtime :: Line Downtime State Time Stamp
Equipment/Line/Downtime :: Line Downtime Equipment Name
Equipment/Line/Downtime :: Line Downtime Reason Split
Equipment/Line/Downtime :: Line Downtime Reason
Equipment/Line/Downtime :: Line Downtime Note
Equipment/Line/Downtime :: Line Downtime Equipment Path
Equipment/Line/Downtime :: Line Downtime Event Sequence
Equipment/Line/Downtime :: Line Downtime Reason Path
```

system.mes.analysis.getMESAnalysisSettings

Description

Get the specified stored analysis settings.

Syntax

system.mes.analysis.getMESAnalysisSettings(savedSettingsName)

- Parameters

String savedSettingsName - The name of the saved analysis settings to return.

- Returns

MESAnalysisSettings - A MESAnalysisSettings object that contains the data points, filter expressions, group by, order by, etc.

- Scope

All

Code Examples

Code Snippet

```
system.mes.analysis.getMESAnalysisSettings('report')
```

Output

```
[report]
AnalysisSettings (250dd58f-554a-493a-9b7e-9df905fc1d29,
report, 0 parents, 0 children, 0 custom properties, 2 complex
properties)
```

system.mes.analysis.getOrderByOptions

Description

Return order by options that can be used when executing analysis.

Syntax

system.mes.analysis.getOrderByOptions(groupFilter, itemFilter)

- Parameters

String groupFilter - A filter to limit the order by options returned to one or more groups. Multiple groups can be specified by separating them with commas.

String itemFilter - A filter to limit the order by options returned to one or more items. Multiple order by items can be specified by separating them with commas.

- Returns

List<AbstractValueItemInfo> - Returns a map containing the filter group path in the key and a list of AbstractValueItemInfo objects in the value. See AbstractValueItemInfo object documentation for details.

- Scope

All

Code Examples

Code Snippet

```
#Get orderby options
options = system.mes.analysis.getOrderByOptions('OEE','')
for item in options:
    for x in options[item]:
        print item, '::', x.getName()
```

Output

```
OEE :: OEE Infeed Count Equipment Path
OEE :: OEE Outfeed Count
OEE :: Target Changeover Time
OEE :: Runtime
OEE :: OEE
OEE :: Standard Rate
OEE :: OEE Reject Count
OEE :: Short Stop Time
OEE :: OEE Infeed Count
OEE :: Elapsed Time
OEE :: Planned Downtime
OEE :: OEE Outfeed Count Equipment Path
OEE :: Unplanned Downtime
OEE :: OEE General Count
```

Overview

These script functions are used to clear the MES analysis cache. If the equipmentPath parameter is not provided, all of the equipment caches are cleared. If the equipmentPath parameter is provided, then only the corresponding equipment cache will be cleared.

Method Options

system.mes.analysis.invalidateAnalysisCache(equipmentPath)

Description

Clears the MES analysis cache. If the equipmentPath parameter is not provided, all of the equipment caches are cleared. If the equipmentPath parameter is provided, then only the corresponding equipment cache will be cleared.

Syntax

system.mes.analysis.invalidateAnalysisCache(equipmentPath)

- Parameters

String equipmentPath - Optional path to the equipment that the cache will be cleared.

- Returns

None

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Nuts Unlimited\Folsom\Mixing\Mixing Line 1'
system.mes.analysis.invalidateAnalysisCache(eqPath)
```

system.mes.analysis.invalidateAnalysisCache()

Description

Clears the MES analysis cache. If the equipmentPath parameter is not provided, all of the equipment caches are cleared. If the equipmentPath parameter is provided, then only the corresponding equipment cache will be cleared.

Syntax

system.mes.analysis.invalidateAnalysisCache()

- Parameters

None

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet

system.mes.analysis.invalidateAnalysisCache()
```

system.mes.analysis.saveMESAnalysisSettings

Description

Save the analysis settings to storage.

Syntax

system.mes.analysis.saveMESAnalysisSettings(settings)

- Parameters

MESAnalysisSettings settings - A MESAnalysisSettings object containing the data points, filter expressions, group by, order by, etc. to save.

- Returns

None

- Scope

All

Code Examples

```
Code Snippet

#Save the settings after creating an analysis settings
settings = system.mes.analysis.createMESAnalysisSettings('Analy
sis')
system.mes.analysis.saveMESAnalysisSettings(settings)
```

system.mes.analysis.getMESAnalysisSettingsList

Description

Return a list of names of stored analysis settings.

Syntax

system.mes.analysis.getMESAnalysisSettingsList()

- Parameters

None

- Returns

List<String> - A list object containing strings of the stored analysis names.

- Scope

All

Code Examples

Code Snippet

```
system.mes.analysis.getMESAnalysisSettingsList()
```

Output

```
 [analysis, lineanalysis, Production Data, rrt]
```

9.7.4 system.mes.oee

Available system.mes.oee functions

system.mes.oee.abortRun

Description

Abort the changeover or production segment that is currently running at the specified equipment. If multiple operations are running at the specified equipment, the last one started will be aborted.

Syntax

system.mes.oee.abortRun(equipmentPath)

- Parameters

String equipmentPath - The equipment path to abort the production segment for.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
eqPath='[global]\Nuts Unlimited\Folsom\Mixing\Mixing Line 1'
system.mes.oee.abortRun(eqPath)
```

Overview

These script functions are used to begin an OEE operation.

Method Options

system.mes.oee.beginOEERun(operationsRequestLink)

Description

Begin an OEE operation for the specified operationsRequestLink. The operations objects must have previously been created prior to calling this function.

Syntax

system.mes.oee.beginOEERun(operationsRequestLink)

- Parameters

MESObjectLink operationsRequestLink - The MES object link to an Operations Request object to start the OEE operation for. An Operations Request object is created when production has been previously scheduled.

- Returns

MESResponseSegment - The Response Segment object as a result of beginning the OEE run.

- Scope

All

Code Examples

Code Snippet

```
#specify equipment path
eqPath = '[global]\Enterprise\Site\Area\Line 1'
#specify the start and end dates
begin = system.date.now()
end = system.date.addDays(system.date.now(), 5)
#specify the category
category = 'Active'
#get the schedule entries
list = system.mes.getEquipmentScheduleEntries(eqPath, begin,
end, category, False)
print list
for item in list:
    print item.getScheduledStartDate()
    if item.hasMESOperationsScheduleLink():
        print item.getMESOperationsScheduleLink()
        print item.getMESOperationsRequestLink(), ';
Operations Request'
        if item.hasMESOperationsResponseLink():
            print item.getMESOperationsResponseLink(), ';
Operations Response'
        else:
            system.mes.oee.beginOEERun(item.
getMESOperationsRequestLink()) ##begin OEE Run
    break
```

Output

```
Size 3
2017-04-14 11:27:00.0
(type: Operations Schedule,  uuid: 8689710a-71aa-4c1e-8a9a-
50205d34568f)
PC01-Enterprise:Site:Area:Line 1 ; Operations Request
ResponseSegment (ab16662d-a5f4-4864-86a4-ccf1a5e6c6e9, PC01-
Enterprise:Site:Area:Line 1_CO, 0 parents, 0 children, 0
custom properties, 7 complex properties)
```

system.mes.oee.beginOEERun(operationsRequest)

Description

Begin an OEE operation for the specified operationsRequest object. The operations objects must have previously been created prior to calling this function.

Syntax

system.mes.oee.beginOEERun(operationsRequest)

- Parameters

MESOperationsRequest operationsRequest - The Operations Request object to start the OEE operation for. An Operations Request object is created when production has been previously scheduled.

- Returns

MESResponseSegment - The Response Segment object as a result of beginning the OEE run.

- Scope

All

Code Examples

Code Snippet

```
#Get the operations-request object
operationsRequest = system.mes.loadMESObject('61f06e29-79ee-
46d0-b4cb-472a7b7af42b')
system.mes.oee.beginOEERun(operationsRequest)#begin OEE run
```

Output

```
ResponseSegment (0354b360-16c6-4cc3-a4ce-25c11b1a8835, Sugar-
Nuts Unlimited:Site 1:Area:Line 1_CO, 0 parents, 0 children, 0
custom properties, 7 complex properties)
```

system.mes.oee.beginOEERun(materialName, equipmentPath)

Description

Begin an OEE operation for the specified material and equipment. The operations objects must have previously been created prior to calling this function.

Syntax

system.mes.oee.beginOEERun(materialName, equipmentPath)

- Parameters

String materialName - The material name to use when starting the OEE run.

String equipmentPath - The equipment path to start the OEE operation for.

- Returns

MESResponseSegment - The Response Segment object as a result of beginning the OEE run.

- Scope

All

Code Examples

Code Snippet

```
materialName = 'Salt'
path = '[global]\Nuts Unlimited\Site 1\Area\Line 2'
system.mes.oee.beginOEERun(materialName, path)
```

Output

```
 ResponseSegment (5be50944-32ea-436b-b819-2b95433869b2, Sugar-
Nuts Unlimited:Site 1:Area:Line 1_CO, 0 parents, 0 children, 0
custom properties, 7 complex properties)
```

system.mes.oee.beginOEERun(workOrder, materialName, equipmentPath)

Description

Begin an OEE operation for the specified operationsRequestLink. The operations objects must have previously been created prior to calling this function.

Syntax

system.mes.oee.beginOEERun(workOrder, materialName, equipmentPath)

- Parameters

String workOrder - The work order to assign to the OEE run.

String materialName - The material name to use when starting the OEE run.

String equipmentPath - The equipment path to start the OEE operation for.

- Returns

MESResponseSegment - The Response Segment object as a result of beginning the OEE run.

- Scope

All

Code Examples

Code Snippet

```
workOrder = 'Wo90'
materialName = 'Sugar'
path = '[global]\Nuts Unlimited\Site 1\Area\Line 1'
system.mes.oee.beginOEERun(workOrder, materialName, path)
```

Output

```
 ResponseSegment (62d81809-9cba-49e3-9622-6b21f276c219, Sugar-
Nuts Unlimited:Site 1:Area:Line 1_CO, 0 parents, 0 children, 0
custom properties, 7 complex properties)
```

system.mes.oee.createMaterialProcessSegment

Description

For the specified material and equipment, create the operations MES object. The operations MES object consist of the Operations Definition, an Operations Segment for changeover and an Operations Segment for production. Each material and equipment combination will have a set of MES operations object.

Syntax

system.mes.oee.createMaterialProcessSegment(materialLink, equipmentPath)

- Parameters

MESObjectLink materialLink - The MES object link to the material definition to base the operations MES objects on.

String equipmentPath - The equipment path to base the operations MES objects on.

- Returns

MESObjectList - A MESObjectList object containing the new Operations Definition and Operations Segment MES objects.

- Scope

All

Code Examples

Code Snippet

```
#specify equipment path
```

```
eqPath = '[global]\Nuts Unlimited\Folsom\Packaging\Packaging
Line 1'

#Get MES object link
matLink = system.mes.getMESObjectLinkByName('MaterialDef', 'Fan
ta')

#create material process segment
list = system.mes.oee.createMaterialProcessSegment(matLink,
eqPath)
for item in list:
    print item
```

Output

```
OperationsSegment (ba291cfd-1e34-4f6d-aa61-c4414a5e9362, Fanta-
Nuts Unlimited:Folsom:Packaging:Packaging Line 1_CO, 0
parents, 0 children, 0 custom properties, 7 complex properties)
OperationsSegment (657bec9b-3a0e-428e-a05a-1e6f321e961c, Fanta-
Nuts Unlimited:Folsom:Packaging:Packaging Line 1, 0 parents, 0
children, 0 custom properties, 7 complex properties)
```

system.mes.oee.endCellChangeover

Description

End the changeover segment that is currently running at the specified equipment. After the changeover segment is ended, the production segment will begin.

Syntax

system.mes.oee.endCellChangeover(equipmentPath)

- Parameters

String equipmentPath - The equipment path of the cell or cell group to end the changeover segment for.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
eqPath='[global]\Nuts Unlimited\Folsom\Receiving\Nut
Unloading\Cell A'
system.mes.oee.endCellChangeover(eqPath)
```

system.mes.oee.endOEEChangeover

Description

End the changeover segment that is currently running at the specified equipment. After the changeover segment is ended, the production segment will begin.

Syntax

system.mes.oee.endOEEChangeover(equipmentPath)

- Parameters

String equipmentPath - The equipment path to end the changeover segment for.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
eqPath='[global]\Nuts Unlimited\Folsom\Receiving\Nut
Unloading\Cell B'
system.mes.oee.endOEEChangeover(eqPath)
```

system.mes.oee.endOEEProduction

Description

End the production segment that is currently running at the specified equipment. This function is only used if a single operation is running at the specified equipment. If multiple operations are running at the specified equipment, use the indexCellProduct function.

Syntax

system.mes.oee.endOEEProduction(equipmentPath)

- Parameters

String equipmentPath - The equipment path to end the production segment for.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
eqPath='Nuts Unlimited\Folsom\Packaging\Packaging Line 1'
system.mes.oee.endOEEProduction(eqPath)
```

system.mes.oee.getEquipmentAvailableMaterial

Description

Return MES object links for the material definitions that can run at the specified equipment path according to the material name search pattern.

Syntax

system.mes.oee.getEquipmentAvailableMaterial(equipmentPath, searchPattern)

- Parameters

String equipmentPath - The equipment path where the material definitions are run.

String searchPattern - Material name filter pattern to limit the results by. It can contain the * and ? wild card characters.

- Returns

MESList<MESObjectLink> - An MESList containing an MESObjectLink object for each material definition object in the results.

- Scope

All

Code Examples

Code Snippet

```
##For Line 1, get all available materials
path = "Enterprise\\Site\\Area\\Line 1"
list = system.mes.oee.getEquipmentAvailableMaterial(path, '*')
for i in range(list.size()):
    matLink = list.get(i)
    print matLink
```

Output

```
PC_0001
PC_0002
Material X
...
```

system.mes.oee.getMaterialAvailableEquipment

Description

Return MES object links for the equipment that can run the specified material and search pattern.

Syntax

system.mes.oee.getMaterialAvailableEquipment(materialLink, searchPattern)

- Parameters

MESObjectLink materialLink - The MES object link to the material definition to return results for.

String searchPattern - Equipment path filter pattern to limit the results by. It can contain the * and ? wild card characters.

- Returns

MESList<MESObjectLink> - A MESList containing a MESObjectLink object for each equipment object in the results.

- Scope

All

Code Examples

Code Snippet

```
##Get the Equipment available for the Material specified
```

```
##by the supplied MESObjectLink, in this case for material
'PC_0001'
link = system.mes.getMESObjectLinkByName('MaterialDef', 'PC_000
1')
print link
list = system.mes.oee.getMaterialAvailableEquipment(link, 'Ente
rprise\\Site\\*')
for i in range(list.size()):
    obj = list.get(i)
    print obj
```

Output

```
PC_0001
Palletizer
Infeed
...
```

system.mes.oee.getMaterialItems

Description

Return a list of MES object links to material objects for the specified parent. To start at the root material object, set the parentLink parameter to None. Material objects consist of Material Class, Material Def and Material Root type of objects.

Syntax

system.mes.oee.getMaterialItems(parentLink, searchPattern)

- Parameters

MESObjectLink parentLink - The MES object link of the parent of the children to include in the results.

String searchPattern - Child material object name filter pattern to limit the results by. It can contain the * and ? wild card characters.

- Returns

MESList<MESObjectLink> - A MESList containing a MESObjectLink object for each material object in the results.

- Scope

All

Code Examples

Code Snippet

```
##Get all the children (both classes and material definitions)
that are under the Material Root.
##That is, get all material items available for OEE operations.
##The search pattern '*' returns all, but a specific name will
return just the relevant link(s).
matRoot = system.mes.getMESObjectLinkByName('MaterialRoot', 'Ma
terial Root')
list = system.mes.oee.getMaterialItems(matRoot, '*')
for i in range(list.size()):
    matLink = list.get(i)
    print matLink
```

Output

```
Class1
PC_0001
PC_0002
ProductClass
...
```

system.mes.oee.getMaterialOperationSegments

Description

Return a list of Operations Segments that support the specified material.

Syntax

system.mes.oee.getMaterialOperationSegments(materialLink, searchPattern)

- Parameters

MESObjectLink materialLink - The MES object link to the material definition to return results for.

String searchPattern - Operations Segment name filter pattern to limit the results by. It can contain the * and ? wild card characters.

- Returns

MESObjectList - MESObjectList containing Operations Segment objects in the results.

- Scope

All

Code Examples

Code Snippet

```
##Get the Material Class Link
matClassLink = system.mes.getMESObjectLinkByName('MaterialClass
', 'Test Material')
##Get the Material Definition of interest
item = system.mes.oee.getMaterialItems(matClassLink, 'PC_0001')
.get(0)
##Get the List of Material Operation Segments and print each
one out
list = system.mes.oee.getMaterialOperationSegments(item, '*')
print list
for i in range(list.size()):
    obj = list.get(i)
    print obj
```

Output

```
OperationsSegment (d88a7df5-677b-4639-a678-fb999589892e,
PC_0001-Enterprise:Site:Area:Line 1, 0 parents, 0 children, 0
custom properties, 7 complex properties)
OperationsSegment (889efb28-cecd-49e5-ab95-8809b2bd08a4,
PC_0001-Enterprise:Site:Area:Line 1:Code Dater, 0 parents, 0
children, 0 custom properties, 7 complex properties)
```

```
OperationsSegment (daad7ffd-31f2-4844-bc6d-ea238cce6b33,
PC_0001-Enterprise:Site:Area:Line 1:Code Dater_CO, 0 parents,
0 children, 0 custom properties, 7 complex properties)
OperationsSegment (ae4a8f2d-540a-4132-87cb-252871b92fdf,
PC_0001-Enterprise:Site:Area:Line 1:Filler, 0 parents, 0
children, 0 custom properties, 7 complex properties)
OperationsSegment (4be5ac58-1b6b-400b-8ca1-e978a329bab4,
PC_0001-Enterprise:Site:Area:Line 1:Filler_CO, 0 parents, 0
children, 0 custom properties, 7 complex properties)
...
```

system.mes.oee.getOEEActiveSegment

Description

Return the active Response Segment object that is currently running at the specified equipment. This can the changeover or production Response Segment. If multiple operations are running at the specified equipment, the last one started will be returned.

Syntax

system.mes.oee.getOEEActiveSegment(equipmentPath)

- Parameters

String equipmentPath - The equipment path to return the Response Segment object for.

- Returns

MESResponseSegment - The active Response Segment object.

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Nuts Unlimited\Folsom\Receiving\Line 1'
```

```
seg = system.mes.oee.getOEEActiveSegment(eqPath)
print seg
```

Output

```
ResponseSegment (b6a02e78-c667-4a15-aa9a-5720c55eeaad, Receive
Nuts, 0 parents, 0 children, 0 custom properties, 7 complex
properties)
```

system.mes.oee.getOEEAllActiveSegments

Description

Returns MES object links for all of the active Response Segment objects that are currently running at the specified equipment.

Syntax

system.mes.oee.getOEEAllActiveSegments(equipmentPath)

- Parameters

String equipmentPath - The equipment path to return the Response Segment objects for.

- Returns

MESList<MESObjectLink> - A MESList containing a MESObjectLink object for each Response Segment object in the results.

- Scope

All

Code Examples

Code Snippet

```
#Following code snippet gets all active segments for
Production line 4 and will abort them all.
#The segments could consist of Changover and Production
segments

eqPath = '[global]\Enterprise\Site 1\Packaging\Line 4'
sl = system.mes.oee.getOEEAllActiveSegments(eqPath)

for seg in sl:    #Look at each MESObjectLink in the MES
Object list
    print seg    #Name of the Response Segment
    objSeg = seg.getMESObject()    #seg is an MESObjectLink,
objSeg is the actual MESObject
    print type(objSeg)    #helper function if you're not sure
what type of object you are dealing with

    system.mes.abortSegment(objSeg)    #Abort all active
segments on this line
```

Output

```
PC_0001-Enterprise:Site 1:Packaging:Line 4
<type 'com.sepasoft.production.common.model.mesobject.objects.
segment.MESResponseSegment'>
```

system.mes.oee.indexCellProduct

Description

Whenever there are multiple products running on a production line, this script function will move the newest product forward to the passed in cell. It achieves this by searching backwards from the designated cell for a cell indexed to a product with a different operationUUID from the product on the designated cell. If a previous product is discovered, it is indexed forward to the designated cell. If a previous product is not discovered, an error is thrown. In the case that we successfully index to the last cell on the line, the run for the previous product at that cell is ended automatically. Finally, an error is thrown if following through with the indexing operation would completely overwrite a product on the line because it has not yet been indexed further down the line.

Syntax

system.mes.oee.indexCellProduct(equipmentPath, skipChangeover)

- Parameters

String equipmentPath - The equipment path of the cell or cell group to index to the next product.

Boolean skipChangeover - If true, the changeover segment for the cell or cell group will be skipped and production will start immediately.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
eqPath='[global]\Nuts Unlimited\Folsom\Receiving\Nut
Unloading\New Cell Group'
system.mes.oee.indexCellProduct(eqPath, True)
```

system.mes.oee.isPreviousProductIndexed

Description

This function checks whether or not a production line is ready to start a new run, especially in the context where multiple products are being run on a line simultaneously. It checks whether or not the previous product on the line has been indexed to the next cell so that the first cell in the line is open for the new product. If the previous product has not been indexed beyond the first cell, the line is not ready to run a new product.

Syntax

system.mes.oee.isPreviousProductIndexed(equipmentPath)

- Parameters

String equipmentPath - The equipment path of the line to check the indexing status.

- Returns

Boolean - True if the previous product has been indexed and is not in changeover, False otherwise.

- Scope

All

Code Examples

Code Snippet

```
eqPath = '[global]\Nuts Unlimited\Folsom\Receiving\Line 1'
indexed = system.mes.oee.isPreviousProductIndexed(eqPath)
print indexed
```

Output

```
True
```

system.mes.oee.removeMaterialOperationSegments

Description

Remove the existing operations MES objects for the specified material and equipment. Each material and equipment combination will have a set of MES operations objects.

Syntax

system.mes.oee.removeMaterialOperationSegments(materialLink, equipmentPath)

- Parameters

MESObjectLink materialLink - The MES object link to the material definition to remove the operations MES objects.

String equipmentPath - The equipment path to remove the operations MES objects.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
path = "Enterprise\\Site\\Area\\Line 1\\Cell Group\\Work Cell"
matRoot = system.mes.getMESObjectLinkByName('MaterialClass', 'T
est Material')
matLink = system.mes.oee.getMaterialItems(matRoot, 'PC_0001').
get(0)
system.mes.oee.removeMaterialOperationSegments(matLink, path)
```

Output

system.mes.oee.updateMaterialOperationSegments

Description

Update the operations MES objects for the specified material and equipment.

Syntax

system.mes.oee.updateMaterialOperationSegments(materialLink, equipmentPath, operationSegmentList)

- Parameters

MESObjectLink materialLink - The MES object link to the material definition that the operations MES objects are based on.

String equipmentPath **-** The equipment path that the operations MES objects are based on.

MESObjectList operationSegmentList - A MESObjectList that contain an Operations Definition, an Operations Segment for the changeover and an Operations Segment for the production.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
path = "DPSG\\Northlake\\Packaging\\Line 5"
matRoot = system.mes.getMESObjectLinkByName('MaterialClass', 'T
est Material')
print matRoot
matLink = system.mes.oee.getMaterialItems(matRoot, 'PC_0001').
get(0)
print matLink

##Get Operations Def
obj1 = system.mes.loadMESObject('PC_0001-DPSG:Northlake:
Packaging:Line 5', 'OperationsDefinition')
##Get Operations Segment for Changeover
obj2 = system.mes.loadMESObject('PC_0001-DPSG:Northlake:
Packaging:Line 5_CO', 'OperationsSegment')
##Get Operations Segment for Production
obj3 = system.mes.loadMESObject('PC_0001-DPSG:Northlake:
Packaging:Line 5', 'OperationsSegment')

productionSettings = obj3.getComplexProperty('ProductionSetting
s', 0)
```

```
##Response Segment Production Settings Complex Property Set
Functions:
#    productionSettings.setEquipmentRefUUID(String
equipmentRefUUID)
#    productionSettings.setEquipmentRefType(String
equipmentRefType)
#    productionSettings.setEquipmentRef(MESObjectLink
mesObjectLink)
#    productionSettings.setModeRefUUID(String modeRefUUID)
#    productionSettings.setModeRefType(String modeRefType)
#    productionSettings.setModeRef(MESObjectLink mesObjectLink)
##Use the OEE rate for the line as an example property to set:
productionSettings.setOEERate(25.0)
obj3.setPropertyValue('ProductionSettings', productionSettings)
system.mes.saveMESObject(obj3)



##Create, populate, and save the list
objList = system.mes.object.list.createList()
objList.add(obj1)
objList.add(obj2)
objList.add(obj3)
system.mes.saveMESObjects(objList)
##Update the Operations Segments
system.mes.oee.updateMaterialOperationSegments(matLink, path,
objList)
```

Output

```
Test Material
Size 3
True
True
True
True
```

9.7.5 system.mes.workorder

Available system.mes.workorder functions

system.mes.workorder.createMESWorkOrder

Description

Creates a work order.

Unique Name

Work orders must be uniquely named. Duplicate names are not allowed.

Syntax

system.mes.workorder.createMESWorkOrder(workOrderName, materialLink)

- Parameters

String workOrderName - Name of the work order to be created.

MESObjectLink materialLink - A MES Object Link to a valid material definition to associate to the work order.

- Returns

MESWorkOrder - A new instance of a MESWorkOrder object.

- Scope

All

Code Examples

Code Snippet

```
##Given a work order name, create the work order and then save
the work order to manifest the change.
matLink = system.mes.getMESObjectLinkByName('MaterialDef', 'Fan
ta')
woObj=system.mes.workorder.createMESWorkOrder('7878', matLink)
system.mes.saveMESObject(woObj)
```

```
print woObj
```

Output

```
WorkOrder (1c397a0d-6ae5-4591-8a54-a6278041b72a, 7878, 0
parents, 0 children, 0 custom properties, 0 complex properties)
```

system.mes.workorder.createMESWorkOrderFilter

Description

Creates a work order filter.

Syntax

system.mes.workorder.createMESWorkOrderFilter()

- Parameters

None

- Returns

MESWorkOrderFilter - A new instance of a MESWorkOrderFilter object.

- Scope

All

Code Examples

Code Snippet

```
##Create a work order filter based on a work order name.
woName = "0752665525"
woFilter = system.mes.workorder.createMESWorkOrderFilter()
woFilter.setWorkOrderNameFilter(woName)
```

```
results = system.mes.workorder.getMESWorkOrderObjectLinkList
(woFilter)
print results
for result in results:
    print result
```

Output

```
Size 1
0752665525
```

system.mes.workorder.deleteMESWorkOrder

Description

Deletes a work order.

Syntax

system.mes.workorder.deleteMESWorkOrder(workOrderName)

- Parameters

String workOrderName - The name of work order to be deleted.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
system.mes.workorder.deleteMESWorkOrder(woName)
```

system.mes.workorder.getMESWorkOrder

Description

Gets the work order object.

Syntax

system.mes.workorder.getMESWorkOrder(workOrderName)

- Parameters

String workOrderName - The name of a work order.

- Returns

MESWorkOrder - A MESWorkOrder object.

- Scope

All

Code Examples

Code Snippet

```
##Get a work order, set the quantity, and save the work order
to manifest the change.
woObj = system.mes.workorder.getMESWorkOrder('0752665525')
woObj.setWorkOrderQuantity(float(6754))
system.mes.workorder.saveMESWorkOrder(woObj)
print woObj
```

Output

```
WorkOrder (a289e509-656b-4914-a597-386e7cf7376b, 0752665525, 0
parents, 0 children, 0 custom properties, 0 complex
properties)
```

system.mes.workorder.getMESWorkOrderObjectLinkList

Description

Get a MESObjectLink list of work orders.

Syntax

system.mes.workorder.getMESWorkOrderObjectLinkList(workOrderFilter)

- Parameters

MESWorkOrderFilter workOrderFilter - A work order filter.

- Returns

MESList<MESObjectLink> - A list of MESObjectLink objects.

- Scope

All

Code Examples

Code Snippet

```
woName = "0752665525"
woFilter = system.mes.workorder.createMESWorkOrderFilter()
woFilter.setWorkOrderNameFilter(woName)
results = system.mes.workorder.getMESWorkOrderObjectLinkList
(woFilter)
print results
for result in results:
    print result
```

Output

```
Size 1
0752665525
```

system.mes.workorder.getMESWorkOrders

Description

Gets a list of work orders.

Syntax

system.mes.workorder.getMESWorkOrders(workOrderFilter)

- Parameters

MESWorkOrderFilter workOrderFilter - A work order filter.

- Returns

List<MESWorkOrder> - A list of MESWorkOrder objects.

- Scope

All

Code Examples

Code Snippet

```
##Create a work order filter.  Get work orders based on the
filter.
##Print the list and the work order object in the list.
woName = "0752665525"
woFilter = system.mes.workorder.createMESWorkOrderFilter()
woFilter.setWorkOrderNameFilter(woName)
results = system.mes.workorder.getMESWorkOrders(woFilter)
print results
for result in results:
```

```
    print result
```

Output

```
Size 1
WorkOrder (a289e509-656b-4914-a597-386e7cf7376b, 0752665525, 0
parents, 0 children, 0 custom properties, 0 complex
properties)
```

system.mes.workorder.saveMESWorkOrder

Description

Save a work order. This is necessary after creating the work order itself, or changing one of it's properties.

Syntax

system.mes.workorder.saveMESWorkOrder(workOrder)

- Parameters

MESWorkOrder workOrder - The work order to be saved.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
##Get a work order, set the quantity, and save the work order
to manifest the change.
```

```
woObj = system.mes.workorder.getMESWorkOrder('0752665525')
woObj.setWorkOrderQuantity(float(6754))
system.mes.workorder.saveMESWorkOrder(woObj)
print woObj
```

Output

```
WorkOrder (a289e509-656b-4914-a597-386e7cf7376b, 0752665525, 0
parents, 0 children, 0 custom properties, 0 complex
properties)
```

9.7.6 system.quality.spc

system.quality.spc.controllimit.calcControlLimitValue

Info

Control limits normally are calculated using the control charts components and when the process is determined to be stable. In cases where additional flexibility is required, this scripting function is provided to calculate control limits from data provided in the parameters. Control limit values for a specified location, sample definition (test), attribute and control limit can be calculated by calling this function. The control limit will be calculated using the control limit configured in the designer and the data specified in the parameters. To set the actual control limit value, use the setControlLimitValue function with the result from this function.

system.quality.spc.controllimit.calcControlLimitValue(locationPath, definition, attributeName, limitName, data)

Description

This script function is used to calculate the control limit value.

Syntax

system.quality.spc.controllimit.calcControlLimitValue(locationPath, definition, attributeName, limitName, data)

- Parameters

String locationPath - The full path of the location to set the control limit. Optionally, it can be left blank to set the default control limit value that is not tied to any location.

String definition - Sample definition to the control limit for.

String attributeName - Name of the attribute within the definition to set the control limit for.

String limitName - Name of the control limit to set.

Dataset data - A dataset containing SPC results to calculate the control limit from.

- Returns

A reference to the results containing the calculated control limit and any messages. See Control Limit Calculated Value for more information.

- Scope

All

Code Examples

Code Snippet

Output

```
Remove this if it the snippet doesn't include print statements
```

system.quality.spc.controllimit.calcControlLimitValue(locationPath, definition, attributeName, limitName, from, to)

Description

This script function is used to calculate the control limit value.

Syntax

system.quality.spc.controllimit.calcControlLimitValue(locationPath, definition, attributeName, limitName, from, to)

- Parameters

String locationPath - The full path of the location to set the control limit. Optionally, it can be left blank to set the default control limit value that is not tied to any location.

String definition - Sample definition to the control limit for.

String attributeName - Name of the attribute within the definition to set the control limit for.

String limitName - Name of the control limit to set.

Date from - Calculate the control with data starting with this date.

Date to - Calculate the control with data ending with this date.

- Returns

A reference to the results containing the calculated control limit and any messages. See Control Limit Calculated Value for more information.

- Scope

All

Code Examples

Code Snippet

```
#This is a sample client script to change a control limit to a
fixed value.
#Define the starting date to calculate the control limit
from java.util import Calendar
fromDate = Calendar.getInstance();
fromDate.add(Calendar.DAY_OF_MONTH, -1)
#Define the endingdate to calculate the control limit
toDate = Calendar.getInstance();
#Get the sample definition based on its name
sampleDef = system.quality.definition.getSampleDefinition('SQLT
ag-Line 1 Checkweigher')
#Calculate the new control limit value
result = system.quality.spc.controllimit.calcControlLimitValue(
'New Enterprise\New Site\Packaging\Line 1\Line 1 Quality',
sampleDef, 'Weight', 'Individual LCL', fromDate.getTime(),
toDate.getTime())
```

```
#Check the results to make sure there are no messages
if result != None and result.hasMessage() == 0:
#Set the actual control limit to the new calculated value
system.quality.spc.controllimit.setControlLimitValue('New
Enterprise\New Site\Packaging\Line 1\Line 1 Quality',
sampleDef, 'Weight', 'Individual LCL', result.
getCalculatedValue())
```

system.quality.spc.controllimit.getLimitNameList

Description

Return a list of names of the defined control limits.

The version of this method that require project name as parameter is deprecated and the version that doesn't require the project name should be used.

Syntax

system.quality.spc.controllimit.getLimitNameList()

- Parameters

None

- Returns

List - An instance of a java List containing the control limit names as strings.

- Scope

All

Code Examples

Code Snippet

```
Output
```

system.quality.spc.controllimit.isValueWithinLimitsByDefName

See the Tech note: isValueWithinLimits

Description

Tests a measurement value to determine if it is within control limits.

Syntax

system.quality.spc.controllimit.isValueWithinLimitsByDefName(locationPath, defName, attributeName, limitNames, productCode, value)

- Parameters

String locationPath - The path of the location that the limit values are begin used to test the measurement value.

String defName - The SPC sample definition name that contains the limit values used in the test.

String attributeName - The name of the attribute that the limit values are used for the test.

String limitNames - The name of the limits to test. Multiple limit names can be specified by separating them with commas.

String productCode - The product code that the limit values used in the test. Use a blank string to specify the default product code. This only applies if the Save Control Limits by Product Code option is selected.

Double value - The measurement value to test.

- Returns

List<ControlLimitResult> - An instance of a java List containing ControlLimitResult objects.

- Scope

All

system.quality.spc.controllimit.isValueWithinLimitsByDefUUID

See the Tech note: isValueWithinLimits

Description

Tests a measurement value to determine if it is within control limits.

The version of this method that require project name as parameter is deprecated and the version that doesn't require the project name should be used.

Syntax

system.quality.spc.controllimit.isValueWithinLimitsByDefUUID(locationPath, defUUID, attributeName, limitNames, productCode, value)

- Parameters

String locationPath - The path of the location that the limit values are begin used to test the measurement value.

String defUUID - The SPC sample definition UUID that contains the limit values used in the test.

String attributeName - The name of the attribute that the limit values are used for the test.

String limitNames - The name of the limits to test. Multiple limit names can be specified by separating them with commas.

String productCode - The product code that the limit values used in the test. Use a blank string to specify the default product code. This only applies if the Save Control Limits by Product Code option is selected.

Double value - The measurement value to test.

- Returns

List<ControlLimitResult> - A java List containing ControlLimitResult objects.

- Scope

All

system.quality.spc.controllimit.removeControlLimitValue

system.quality.spc.controllimit.removeControlLimitValue(locationPath, definition, attributeName, limitName, productCode)

Description

Remove a previously set control limit value. This only applies if the Save Control Limits by Product Code option is selected.

Syntax

system.quality.spc.controllimit.removeControlLimitValue(locationPath, definition, attributeName, limitName, productCode)

- Parameters

String locationPath - The path of the location that the limit value is being removed. If this is a blank string, then all product code control limit values for all locations will be removed.

SampleDefinition definition - The SPC sample definition object that contains the limit value being removed.

String attributeName - The name of the attribute that the limit value is being removed. If this is a blank string, then all product code control limits for all attributes will be removed.

String limitName - The name of the limit to remove the value. If this is a blank string, then all product code control limit values will be removed.

String locationPath - The product code that the limit value is being removed. If this parameter is not specified, then all product code control limit values will be removed.

String productCode - Name of the product code.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet
```

```
Output
```

system.quality.spc.controllimit.removeControlLimitValue(locationPath, definition, attributeName, limitName)

Description

Remove a previously set control limit value. This only applies if the Save Control Limits by Product Code option is selected.

Syntax

system.quality.spc.controllimit.removeControlLimitValue(locationPath, definition, attributeName, limitName)

- Parameters

String locationPath - The path of the location that the limit value is being removed. If this is a blank string, then all product code control limit values for all locations will be removed.

SampleDefinition definition - The SPC sample definition object that contains the limit value being removed.

String attributeName - The name of the attribute that the limit value is being removed. If this is a blank string, then all product code control limits for all attributes will be removed.

String limitName - The name of the limit to remove the value. If this is a blank string, then all product code control limit values will be removed.

String locationPath - The product code that the limit value is being removed. If this parameter is not specified, then all product code control limit values will be removed.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

Output

system.quality.spc.controllimit.setControlLimitValue

system.quality.spc.controllimit.setControlLimitValue(locationPath, definition, attributeName, limitName, value)

Description

Control limits normally are set using the control charts components and when the process is determined to be stable. In cases where additional flexibility is required, this scripting function is provided. New control limit values for a specified location, sample definition (test), attribute and control limit can be set by calling this function.

Syntax

system.quality.spc.controllimit.setControlLimitValue(locationPath, definition, attributeName, limitName, value)

- Parameters

String locationPath - The full path of the location to set the control limit. Optionally, it can be left blank to set the default control limit value that is not tied to any location.

Sample Definition definition - Sample definition to the control limit for.

String attributeName - Name of the attribute within the definition to set the control limit for.

String limitName - Name of the control limit to set.

Double value - New control limit value.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#This is a sample client script to change a control limit to a
fixed value.
sampleDef = system.quality.definition.getSampleDefinition("Samp
leDefName")
system.quality.spc.controllimit.setControlLimitValue('New
Enterprise\New Site\Packaging\Line 1\Line 1 Quality',
sampleDef, 'Weight', 'Individual LCL', 100.0)
```

system.quality.spc.controllimit.setControlLimitValue(locationPath, definition, attributeName, limitName, productCode, value)

Description

Control limits normally are set using the control charts components and when the process is determined to be stable. In cases where additional flexibility is required, this scripting function is provided. New control limit values for a specified location, sample definition (test), attribute and control limit can be set by calling this function.

Save Control Limits by Product Code should be set at the Enterprise level of the Production Model to take advantage of the functionality.

Syntax

system.quality.spc.controllimit.setControlLimitValue(locationPath, definition, attributeName, limitName, productCode, value)

- Parameters

String locationPath - The full path of the location to set the control limit. Optionally, it can be left blank to set the default control limit value that is not tied to any location.

Sample Definition definition - Sample definition to the control limit for.

String attributeName - Name of the attribute within the definition to set the control limit for.

String limitName - Name of the control limit to set.

String productCode - Name of the product code to set.

Double value - New control limit value.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#This is a sample client script to change a control limit to a
fixed value.
sampleDef = system.quality.definition.getSampleDefinition("Samp
leDefName")
system.quality.spc.controllimit.setControlLimitValue('New
Enterprise\New Site\Packaging\Line 1\Line 1 Quality',
sampleDef, 'Weight', 'Individual LCL', 'PC
_01', 100.0)
```

system.quality.spc.deleteStoredSPC

Description

Delete the SPC settings for specified saved SPC settings name.

Syntax

system.quality.spc.deleteStoredSPC(name)

- Parameters

String name - The name of the stored SPC settings to delete.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

Output

Method Options:

system.quality.spc.exportSPCResults(settings, locale, csvSeparator)

Description

Export SPC data to a CSV formatted string.

Syntax

system.quality.spc.exportSPCResults(settings, locale, csvSeparator)

- Parameters

SPCSettings settings - An instance of a SPCSettings object to base the export results. Use the system.quality.spc.settings.createSettings script function to create the SPCSettings object.

String locale - The locale name to used for formatting numeric values.

String csvSeparator - The character to use as the value separator.

- Returns

A string containing the exported SPC data in CSV format.

- Scope

All

Code Examples

Code Snippet

Output

system.quality.spc.exportSPCResults(settings, locale, csvSeparator, filter)

Description

Export SPC data to a CSV formatted string.

Syntax

system.quality.spc.exportSPCResults(settings, locale, csvSeparator, filter)

- Parameters

SPCSettings settings - An instance of a SPCSettings object to base the export results. Use the system.quality.spc.settings.createSettings script function to create the SPCSettings object.

String locale - The locale name to used for formatting numeric values.

String csvSeparator - The character to use as the value separator.

String filter - Column names to include in the export separated by commas.

- Returns

A string containing the exported SPC data in CSV format.

- Scope

All

Code Examples

Code Snippet

Output

system.quality.spc.format.fromSPCCategoryTypes

Description

This is intended for internal use. Based on the category of control chart (SPCCategoryTypes) and the data format (SPCDataFomat), return the appropriate data format.

Syntax

system.quality.spc.format.fromSPCCategoryTypes(type, dataFormat)

- Parameters

SPCCategoryTypes type - An instance of a SPCCategoryTypes object that represents the category of control chart.

SPCDataFormat dataFormat - An instance of a SPCDataFormat object that represents the SPC data format.

- Returns

SPCDataFormat - A reference to the matching SPCDataFormat object.

- Scope

All

Code Examples

Code Snippet

Output

system.quality.spc.format.getEnum

system.quality.spc.format.getEnum(value)

Description

Returns SPCDataFormat for the specified ordinal or name value. The SPC data formats represent the control chart types and is used when specifying the kind SPC results to return.

Syntax

system.quality.spc.format.getEnum(value)

- Parameters

Int value - The ordinal value representing the SPCDataFormat.

- Returns

SPCDataFormat - A reference to the matching SPCDataFormat object.

- Scope

All

Code Examples

Code Snippet

Output

```
Remove this if it the snippet doesn't include print statements
```

system.quality.spc.format.getEnum(name)

Description

Returns SPCDataFormat for the specified ordinal or name value. The SPC data formats represent the control chart types and is used when specifying the kind SPC results to return.

Syntax

system.quality.spc.format.getEnum(name)

- Parameters

String name - The ordinal value representing the SPCDataFormat.

- Returns

SPCDataFormat - A reference to the matching SPCDataFormat object.

- Scope

All

Code Examples

Code Snippet

Output

```
Remove this if it the snippet doesn't include print statements
```

system.quality.spc.format.getEnumName

Description

Returns the name of the SPCDataFormat from the specified display name. The display name is the user friendly name where the name cannot contain spaces. The SPC data formats represent the control chart types and is used when specifying the kind SPC results to return.

Syntax

system.quality.spc.format.getEnumName(displayName)

- Parameters

String displayName - The ordinal value representing the SPCDataFormat.

- Returns

String name - The name of the SPCDataFormat.

- Scope

All

Code Examples

Code Snippet

Output

system.quality.spc.format.getSPCCategoryType

Description

This is intended for internal use. Based on the category of control chart (SPCCategoryTypes) and the data format (SPCDataFomat), return the appropriate data format.

Syntax

system.quality.spc.format.getSPCCategoryType(dataFormat)

- Parameters

SPCDataFormat dataFormat - parameterAn instance of a SPCCategoryTypes object that represents the category of control chart.

SPCDataFormat dataFormat - An instance of a SPCDataFormat object that represents the SPC data format.

- Returns

SPCCategoryTypes - A reference to the matching SPCDataFormat object.

- Scope

All

Code Examples

Code Snippet

Output

system.quality.spc.format.getSPCDataFormatAsDataset

Description

Returns an instance of a Dataset object containing the SPC data format options. The SPC data formats represent the control chart types and is used when specifying the kind SPC results to return.

Syntax

system.quality.spc.format.getSPCDataFormatAsDataset()

- Parameters

None

- Returns

Dataset - A new instance of a Dataset object containing the SPC data format options.

- Scope

All

Code Examples

```
Code Snippet
```

```
Output
```

system.quality.spc.format.valueOf

Description

Syntax

system.quality.spc.format.valueOf()

- Parameters

Type name - description

- Returns

Type - description

- Scope

All

Code Examples

Code Snippet

Output

```
Remove this if it the snippet doesn't include print statements
```

system.quality.spc.format.values

Description

Syntax

system.quality.spc.format.values()

- Parameters

Type name - description

- Returns

Type - description

- Scope

All

Code Examples

Code Snippet

Output

```
Remove this if it the snippet doesn't include print statements
```

system.quality.spc.getSPCResults

Description

Returns SPC results for the specified settings.

Syntax

getSPCResults(settings)

- Parameters

SPCSettings settings - An instance of a SPCSettings object to base the results. Use the system.quality.spc.settings.createSettings script function to create the SPCSettings object.

- Returns

An instance of a SPCResults object containing the SPC data.

- Scope

All

Code Examples

```
Code Snippet
```

system.quality.spc.getSPCStoredResults

Description

Returns SPC results for the specified stored SPC settings and date range.

Syntax

system.quality.spc.getSPCStoredResults(name, fromDate, toDate)

- Parameters

String name - The name of the stored SPC settings to base the results.

Date fromDate - The start of the date range to include in the results.

Date toDate - The end of the date range to include in the results.

- Returns

SPCResults results - An instance of a SPCResults object containing the SPC data.

- Scope

All

Code Examples

```
Code Snippet
```

```
Output
```

system.quality.spc.loadStoredSPC

Description

Returns the SPC settings for the specified saved SPC settings name.

Syntax

system.quality.spc.loadStoredSPC(name)

- Parameters

String name - The name of the stored SPC settings.

- Returns

SPCSettings - An instance of a SPCSettings object.

- Scope

All

Code Examples

```
Code Snippet
```

```
Output
```

system.quality.spc.renameStoredSPC

Description

Rename the SPC settings for specified saved SPC settings name.

Syntax

system.quality.spc.renameStoredSPC(name, newName)

- Parameters

String name - The existing name of the stored SPC settings to rename.

String newName - The new name of the stored SPC settings.

- Returns

None

- Scope

All

Code Examples

Code Snippet

Output

system.quality.spc.saveStoredSPC

Description

Rename the SPC settings for specified saved SPC settings name.

Syntax

system.quality.spc.saveStoredSPC(name, settings, overwrite)

- Parameters

String name - The name of the stored SPC settings to save.

SPCSettings settings - An instance of a SPCSettings object to save. Use the system.quality.spc.settings.createSettings script function to create the SPCSettings object.

Boolean overwrite - If true and the stored SPC settings already exist, save the new settings over the existing settings.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

Output

system.quality.spc.settings.createSettings

Description

Create a new instance of a SPCSettings object based on the parameters.

Syntax

system.quality.spc.settings.createSettings(definitionName, attribute, filters, controlLimits, signals, dataFormatName)

- Parameters

String definitionName - The sample definition name for the new settings.

String attribute - The attribute name for the new settings.

String filters - The filters for the new settings. Multiple filter expressions can be separated by commas.

String controlLimits - The control limits for the new settings. Multiple control limits can be separated by commas.

String signals - The SPC rules (signals) for the new settings. Multiple SPC rules can be separated by commas.

String dataFormatName - The data format (Calculation kind types or control chart type) for the new settings.

- Returns

SPCSettings - A new instance of a SPCSettings object.

- Scope

All

Code Examples

```
Code Snippet

filter = 'FromDate=2016-07-25 00:00:00|ToDate=2015-09-15 23:59:
59|Location=New Enterprise\California\Quality\Location A'

#SPC settings object is created manually
settings = system.quality.spc.settings.createSettings('Def 7',
'Level', filter, '', '', 'Anderson-Darling Test')

#The Anderson Darling Test calculation is executed
result = system.quality.sample.data.executeMiscCalculation
(settings, 'Adt')
print 'Ad: ', result.getValue('Ad')
```

system.quality.spc.settings.decodeFilters

Description

Decode a list of SPC filter expressions into a java Map object. Each filter key can have multiple filter values.

Syntax

system.quality.spc.settings.decodeFilters(filterList)

- Parameters

List filterList - An instance of a java List object containing SPC filter expression strings. Example: "Location=Enterprise\Site\Area\Quality Test Station 1, Location=Enterprise\Site\Area\Quality Test Station 2, Product Code=DEF"

- Returns

Map<String, List<String>> An instance of a java Map. The map key is the filter name. For example, "Location" or "Product Code". The value for the key contains a java List object containing all of the filter values. For example, the key "Location" can have the filters values of "Quality Station 1" and "Quality Station 2".

- Scope

All

Code Examples

Code Snippet

```
```

Output

```
Remove this if it the snippet doesn't include print statements
```

system.quality.spc.settings.decodeList

Description

Decode a string that can represent a SPC filter, control limits, SPC rules (signals), etc. into a java List object. The input string will be parsed on wither the comma or pipe (|) character and each parsed result will be added to the returned List object.

Syntax

system.quality.spc.settings.decodeList(input)

- Parameters

String input - The string value to parse.

- Returns

A java List object containing the parsed strings.

- Scope

All

Code Examples

```
Code Snippet
```

```
Output

Remove this if it the snippet doesn't include print statements
```

system.quality.spc.settings.decodeParams

Description

Decode a list of SPC parameters into a java Map object. Each parameter key can have only one parameter value.

Syntax

system.quality.spc.settings.decodeParams(optionalParams)

- Parameters

String optionalParams - A string containing optional parameters separated by either the comma or pipe (|) characters. Example: "PaddingBarCount=4,RowLimit=100, DataBarCount=7,IncludeDisabledAttributes=true"

- Returns

Map<String, String> - An instance of a java Map containing key value pairs.

- Scope

All

Code Examples

```
Code Snippet
```

```
Output

Remove this if it the snippet doesn't include print statements
```

system.quality.spc.settings.encodeList

Description

Encode the specified java List into a single string separated by the pipe (\) character.

Syntax

system.quality.spc.settings.encodeList(list)

- Parameters

String[] list - An instance of a java List object containing string values.

- Returns

A single string containing all of the items from the list.

- Scope

All

Code Examples

```
Code Snippet
```

```
Output

Remove this if it the snippet doesn't include print statements
```

system.quality.spc.settings.formatDate

Description

Returns a string for the specified date to is formatted correctly for a filter expression.

Syntax

system.quality.spc.settings.formatDate()

- Parameters

Date date - A Date object to format.

- Returns

String formattedDate - A SPC settings formatted date string.

- Scope

All

Code Examples

Code Snippet

Output

9.7.7 system.quality.definition

system.quality.definition.addSampleDefinition

Description

Adds the sample definition passed in the parameter to the SPC system. After it has been added it will become available to record samples and for selection on the control charts. Attributes, locations, control limits and signals must be added to the sample definition prior to calling this function. See Sample Definition for more information.

Syntax

system.quality.definition.addSampleDefinition(sampleDefinition)

- Parameters

String sampleDefinition - New sample definition that previously was created in script.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

Output

```
Remove this if it the snippet doesn't include print statements
```

system.quality.definition.attribute.getNew

Description

Creates and returns a new instance of aSampleDefinitionAttributeobject. The new SampleDefinitionAttribute object can be added to a sample definition using the addAttribute method on the sample definition object.

Syntax

system.quality.definition.attribute.getNew()

- Parameters

None

- Returns

A new SampleDefinitionAttribute instance.

- Scope

All

Code Examples

Code Snippet

```
system.quality.definition.attribute.getNew()
```

system.quality.definition.attribute.getSampleDefinitionList

Description

Returns an instance of a Dataset object containing available sample definitions.

Syntax

system.quality.definition.attribute.getSampleDefinitionList(showDisabled, nameFilter, locationPathFilter)

- Parameters

Boolean showDisabled - If true, return only sample definitions that have been disabled.

String nameFilter - Sample definition name filter to limiting the results. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.

String locationPathFilter - Location path filter to limit the results.

- Returns

Dataset - An instance of a Dataset object containing sample definition information.

- Scope

All

Code Examples

Code Snippet

Output

system.quality.definition.attribute.types.dataTypeToType

Description

Return a reference to a SPC attribute data type from an Ignition data type. For example, an Ignition data type of Float8 will be an attribute data type of Real.

Syntax

system.quality.definition.attribute.types.dataTypeToType(dataType)

- Parameters

DataType dataType - Ignition DataType reference that represents the type of data for a sample attribute.

- Returns

A reference to a AttributeDataType value that matches.

- Scope

All

Code Examples

Code Snippet

Output

```
Remove this if it the snippet doesn't include print statements
```

system.quality.definition.attribute.types.intToType

Description

Return a reference to a SPC attribute data type from the ordinal value.

Syntax

system.quality.definition.attribute.types.intToType(ordinal)

- Parameters

Integer ordinal - A valid AttributeDataType ordinal value.

- Returns

A reference to a AttributeDataType value that matches.

- Scope

All

Code Examples

Code Snippet

Output

```
Remove this if it the snippet doesn't include print statements
```

system.quality.definition.getLimitNameList

Description

Return a list of names of the defined control limits.

Syntax

system.quality.definition.getLimitNameList()

- Parameters

None

- Returns

List<String> - An instance of a java List containing the control limit names as strings.

- Scope

All

Code Examples

Code Snippet

Output

system.quality.definition.getNew

Description

Creates and returns a new instance of a SampleDefinition object.

Syntax

system.quality.definition.getNew ()

- Parameters

None

- Returns

A new sample definition instance.

- Scope

All

Code Examples

Code Snippet

Output

```
Remove this if it the snippet doesn't include print statements
```

system.quality.definition.getNewDefinitionAttribute

Description

Create a new instance of a SampleDefinitionAttribute object that can then be added to a sample definition.

Syntax

system.quality.definition.getNewDefinitionAttribute()

- Parameters

None

- Returns

SampleDefinitionAttribute - An instance of a new SampleDefinitionAttribute object.

- Scope

All

Code Examples

```
Code Snippet
```

```
Output
```

system.quality.definition.getNewDefinitionLimit

Description

Create a new instance of a SampleDefinitionControlLimit object that can then be added to a sample definition.

Syntax

system.quality.definition.getNewDefinitionLimit(limitName)

- Parameters

String limitName - Name of the limit to base the new instance.

- Returns

SampleDefinitionControlLimit - An instance of a new SampleDefinitionControlLimit object.

- Scope

All

Code Examples

```
Code Snippet
```

```
Output
```

system.quality.definition.getNewDefinitionLocation

Description

Returns a new sample definition location instance for the production location specified by the locationID parameter. The new instance name is specified by the name parameter. The new SampleDefinitionLocation object can be added to a sample definition using the addAllowedLocation method of a SampleDefinition object.

Syntax

system.quality.definition.getNewDefinitionLocation(locationID, name)

- Parameters

Integer locationID - The location ID from the production model.

String name - The new instance name.

- Returns

SampleDefinitionLocation - A new instance of a SampleDefinitionLocation object.

- Scope

All

Code Examples

```
Code Snippet
```

```
Output
```

system.quality.definition.getNewDefinitionSignal

Description

Create a new instance of a SampleDefinitionSignal object that can then be added to a sample definition.

Syntax

system.quality.definition.getNewDefinitionSignal(signalName)

- Parameters

String signalName - Name of the signal to base the new instance.

- Returns

SampleDefinitionSignal - An instance of a new SampleDefinitionSignal object.

- Scope

All

Code Examples

```
Code Snippet
```

```
Output
```

system.quality.definition.getSampleDefinition

Description

Returns a reference to the sample definition object with a matching ID. The ID is generated by the database when the sample definition was first saved.

Syntax

system.quality.definition.getSampleDefinition(sampleDefID)

- Parameters

Integer sampleDefID - Database created ID for the sample definition.

- Returns

Sample Definition object

- Scope

All

Code Examples

```
Code Snippet
```

```
Output

Remove this if it the snippet doesn't include print statements
```

system.quality.definition.getSampleDefinition(sampleDefName)

Description

Returns a reference to the sample definition object with a matching name. The name is generated by the database when the sample definition was first saved.

Syntax

system.quality.definition.getSampleDefinition(sampleDefName)

- Parameters

String sampleDefName - The name given to the sample definition when it was created.

- Returns

Sample Definition object

- Scope

All

Code Examples

Code Snippet

Output

```
Remove this if it the snippet doesn't include print statements
```

system.quality.definition.getSignalNameList

Description

Return a list of names of the defined signals (SPC rules).

Syntax

system.quality.definition.getSignalNameList()

- Parameters

None

- Returns

List<String> - An instance of a java List containing the signal names as strings.

- Scope

All

Code Examples

Code Snippet

Output

system.quality.definition.location.getNew

Description

Returns a new sample definition location instance for the production location specified by the locationID parameter. The new instance name is specified by the name parameter. The new SampleDefinitionLocation object can be added to a sample definition using the addAllowedLocation method of a SampleDefinition object.

Syntax

system.quality.definition.location.getNew(locationID, name)

- Parameters

int locationID - The location ID from the production model.

String name - The new instance name.

- Returns

A new instance of a SampleDefinitionLocation object.

- Scope

All

Code Examples

Code Snippet

```
system.quality.definition.location.getNew(2, 'packaging line 1'
)
```

system.quality.definition.updateSampleDefinition

Description

Updates an existing sample definition object passed in the parameter. After it has been updated, the changes will be reflected during recording samples and on the control charts.

Syntax

system.quality.definition.updateSampleDefinition (sampleDefinition)

- Parameters

String sampleDefinition - Existing sample definition.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

Output

```
Remove this if it the snippet doesn't include print statements
```

9.7.8 system.quality.sample.data

system.quality.sample.data.approveSample

Description

Approve an existing sample. If the associated sample definition for the specified sample is not set for auto approval, it will have to approved. This can be done using various methods of which this is one of them.

Syntax

system.quality.sample.data.approveSample(sampleUUID, approvedBy)

- Parameters

String sampleUUID - The UUID to an existing sample to approve.

String approvedBy - The name of the person who is approving the sample.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
system.quality.sample.data.approveSample(currentSample.
getSampleUUID, system.security.getUsername())
```

system.quality.sample.data.excludeSample

Description

Excludes the sample specified by uuid parameter.

Syntax

system.quality.sample.data.excludeSample(sampleUUID)

- Parameters

String sampleUUID - The UUID to an existing sample to exclude.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
system.quality.sample.data.excludeSample('88a6b6a4-3177-4759-b594-7220d416c735')
```

system.quality.sample.data.executeMiscCalculation

Description

Perform a previously defined miscellaneous calculation. If calculations other than the built-in calculations, such as PPM, are needed, then they can be defined in the Misc. Calculation section in the MES production model.

See the tech note: Perform Miscellaneous Calculations

Syntax

system.quality.sample.data.executeMiscCalculation(settings, miscCalcName)

- Parameters

SPCSettings settings - An instance of a SPCSettings object that defines the samples to perform the calculation.

String miscCalcName - The name of the miscellaneous calculation, that has been previously defined in the Misc. Calculations, to perform.

- Returns

MiscCalcEvent - The MES object created for the execution of the miscellaneous calculation.

- Scope

All

Code Examples

```
filter = 'FromDate=2015-07-25 00:00:00|ToDate=2015-09-15 23:59:
59|Location=New Enterprise\California\Quality\Location A'

#SPC settings object is created manually
settings = system.quality.spc.settings.createSettings('Def 7',
'Level', filter, '', '', 'Anderson-Darling Test')

#The Anderson Darling Test calculation is executed
result = system.quality.sample.data.executeMiscCalculation
(settings, 'Adt')
print 'Ad: ', result.getValue('Ad')
```

system.quality.sample.data.getCauseList

Description

Return a list of existing assignable causes.

Syntax

system.quality.sample.data.getCauseList()

- Parameters

None

- Returns

List<String> - A list holding assignable causes as strings.

- Scope

All

Code Examples

Code Snippet

```
list = system.quality.sample.data.getCauseList()
for cause in list:
print cause
```

Output

```
 [Prime, New Cause]
```

system.quality.sample.data.getCreateSampleByDefUUID

Description

Return a sample that matches the sampleUUID parameter. If not found, create and return a new sample based on the sample definition that matches the definitionUUID parameter. The newly created sample will also be initialized for the location specified by the locationPath parameter.

Syntax

system.quality.sample.data.getCreateSampleByDefUUID(sampleUUID, defUUID, locationPath)

- Parameters

String sampleUUID - Sample UUID to lookup.

String defUUID - Existing sample definition UUID to base the new sample on.

String locationPath - A valid path to a location to base the new sample for.

- Returns

Sample Object - A reference to the existing sample or the newly created sample.

- Scope

All

Code Examples

Code Snippet

```
sampleUUID = system.gui.getParentWindow(event).
getComponentForPath('Root Container').SampleUUID
locationPath = system.gui.getParentWindow(event).
getComponentForPath('Root Container').LocationPath
#This will return a sample for the sampleUUID. If the
sampleUUID is blank, it will return a new sample
sample = system.quality.sample.data.getCreateSampleByName
(sampleUUID, sampleDef.getDefUUID(), locationPath)
```

system.quality.sample.data.getCreateSampleByName

Description

Return a sample that matches the sampleUUID parameter. If not found, create and return a new sample based on the sample definition that matches the definitionName parameter. The newly created sample will also be initialized for the location specified by the locationPath parameter.

Syntax

system.quality.sample.data.getCreateSampleByName(sampleUUID, defName, locationPath)

- Parameters

String sampleUUID - Sample UUID to lookup.

String defName - Existing sample definition name to base the new sample on.

String locationPath - A valid path to a location to base the new sample for.

- Returns

Sample Object - A reference to the existing sample or the newly created sample.

- Scope

All

Code Examples

Code Snippet

```
sampleUUID = system.gui.getParentWindow(event).
getComponentForPath('Root Container').SampleUUID
locationPath = system.gui.getParentWindow(event).
getComponentForPath('Root Container').LocationPath
```

```
#This will return a sample for the sampleUUID. If the
sampleUUID is blank, it will return a new sample
sample = system.quality.sample.data.getCreateSampleByName
(sampleUUID, 'Viscosity', locationPath)
```

system.quality.sample.data.getNewByDefName

Description

Creates and returns a new sample based on the sample definition that matches the definitionName parameter. The newly created sample will also be initialized for the location specified by the locationPath parameter.

Syntax

system.quality.sample.data.getNewByDefName(defName, locationPath)

- Parameters

String defName - Existing sample definition name to base this sample on.

String locationPath - A valid path to a location.

- Returns

Sample Object - A reference to the newly created sample.

- Scope

All

Code Examples

Code Snippet

```
locationPath = event.source.parent.LocationPath
sampleDefName = event.newValue
sample = system.quality.sample.data.getNewByDefName
(sampleDefName, locationPath)
```

system.quality.sample.data.getNewByDefUUID

Description

Creates and returns a new sample based on the sample definition that matches the defUUID parameter. The newly created sample will also be initialized for the location specified by the locationPath parameter.

Syntax

system.quality.sample.data.getNewByDefUUID(defUUID, locationPath)

- Parameters

String defUUID - Existing sample definition UUID to base this sample on.

String locationPath - A valid path to a location.

- Returns

Sample Object - A reference to the newly created sample.

- Scope

All

Code Examples

Code Snippet

```
locationPath = event.source.parent.LocationPath
defUUID = event.newValue
sample = system.quality.sample.data.getNewByDefUUID(defUUID ,
locationPath)
```

system.quality.sample.data.getSample

Description

Return a sample that matches the sampleUUID parameter.

Syntax

system.quality.sample.data.getSample(sampleUUID)

- Parameters

String sampleUUID - Sample UUID to lookup.

- Returns

Sample Object - A reference to the existing sample.

- Scope

All

Code Examples

Code Snippet

```
sampleUUID = system.gui.getParentWindow(event).
getComponentForPath('Root Container').SampleUUID
sample = system.quality.sample.data.getSample(sampleUUID)
```

system.quality.sample.data.includeSample

Description

Includes the sample specified by uuid parameter.

Syntax

system.quality.sample.data.includeSample(sampleUUID)

- Parameters

String sampleUUID - The UUID to an existing sample to include.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
system.quality.sample.data.includeSample('e462217e-bc7c-4ee6-9226-206a4d7576e8')
```

system.quality.sample.data.purgeSampleData

Description

Purge samples for the specified sample definition UUID that when taken prior to the specified date. The samples will be permanently deleted and cannot be recovered.

Syntax

system.quality.sample.data.purgeSampleData(defUUID, priorToDate)

- Parameters

String defUUID - Existing sample definition UUID to delete the samples for.

Date priorToDate- The cutoff date where only samples entered before will be deleted.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
```

Output

```
Remove this if it the snippet doesn't include print statements
```

system.quality.sample.data.removeSample

Description

Remove a single sample. This function should be used with caution because it permanently removes the data from the database. A sample can be removed at any point in its life cycle. Meaning it can be removed after it has been scheduled but before measurements are recorded and after measurements have been recorded.

Syntax

system.quality.sample.data.removeSample(sampleUUID)

- Parameters

String sampleUUID - The UUID to an existing sample to remove.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet

system.quality.sample.data.removeSample(event.getSampleUUID())
```

system.quality.sample.data.setSampleCause

Description

Set an assignable cause to the specified sample and attribute.

Syntax

system.quality.sample.data.setSampleCause(sample, attributeName, cause, userName)

- Parameters

Sample sample - The sample to add the assignable cause to.

String attributeName - The attribute of the sample to associate the assignable cause to.

String cause - The assignable cause. This can be an existing assignable cause or a new assignable cause. Use the getCauseList() function to return the assignable causes that have previously been used.

String userName - The user name that is adding the assignable cause.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#The following event handler script will set a note and cause
for the sample
sample = event.source.parent.getComponent('Sample Entry').
getSample()
system.quality.sample.data.setsampleNote(sample, 'Attr1', 'This
is a new sample', 'Sarah')
system.quality.sample.data.setSampleCause(sample, 'Attr1', 'New
Cause', 'Sarah')
event.source.parent.getComponent('Sample Entry').save()
```

system.quality.sample.data.setSampleNote

Description

Set a note to the specified sample and attribute.

Syntax

system.quality.sample.data.setSampleNote(sample, attributeName, note, username)

- Parameters

Sample sample - The sample to add the note to.

String attributeName - The attribute of the sample to associate the note to.

String note - The actual note.

String userName - The user name that is adding the note.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#The following event handler script will set a note and cause
for the sample
sample = event.source.parent.getComponent('Sample Entry').
getSample()
system.quality.sample.data.setsampleNote(sample, 'Attr1', 'This
is a new sample', 'Sarah')
system.quality.sample.data.setSampleCause(sample, 'Attr1', 'New
Cause', 'Sarah')
event.source.parent.getComponent('Sample Entry').save()
```

system.quality.sample.data.unapproveSample

Description

Unapprove a previously approved sample. When a sample is unapproved it will not be shown in the control charts or included in the data during automatic signal evaluation.

Syntax

system.quality.sample.data.unapproveSample(sampleUUID)

- Parameters

String sampleUUID - The UUID to an existing sample to unapprove.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet

system.quality.sample.data.removeSample(event.getSampleUUID())
```

system.quality.sample.data.updateSample

Description

Update an existing or new sample. If the valuesRecorded parameter is true, current shift, product code and additional factor information will be recorded along with the measurement values. Because samples may be scheduled, they can be created and updated with no measurement values. This allows for 'coming due', 'due' and 'overdue' functionality to be tracked.

Syntax

system.quality.sample.data.updateSample(locationPath, sample, valuesRecorded)

- Parameters

String locationPath - A valid path to location of the sample to be updated.

String sample - The sample to be updated.

Boolean valuesRecorded - If true, record the values along with the other sample information, including additional factors.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet

 system.quality.sample.data.updateSample('QualityDemo\New
Enterprise\New Site\Packaging\Line 1\Line 1 Quality',
currentSample, 1)
```

9.7.9 system.recipe

system.recipe.addItemToRecipe

Description

Add a production item to a recipe. Once a production item is added to a recipe, the recipe values for the production item can be managed. Also, the recipe can be selected for the added production item.

Syntax

system.recipe. addItemToRecipe (recipeName, itemPath, note)

- Parameters

String recipeName - Name of recipe to add the specified production item.

String itemPath - The item path to a production line, cell, cell group or location.

String note - Optional note to be stored in the recipe change log.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#---- addItemToRecipe ----#
#-- function: system.recipe.addItemToRecipe(recipeName,
itemPath, note)
#                                                (str)
(str)   (str)
#-- optional means that the field can be empty: "" or have a
value
# itemPath can be any line, cell, cell group or location

#-- required arguments
itemPath = '[global]\Enterprise\Site 2\Packaging\Line
1\Holding'
recipeName = "Weinhardt's"

#-- optional arguments
note = ""

#-- execute
try:
    system.recipe.addItemToRecipe(recipeName, itemPath, note)
    print itemPath + ' Added to ' + recipeName
except IOError:
    system.gui.messageBox('Error - this Production Item
already exists for this Recipe', 'Insert Failed')
```

Output

```
[global]\Enterprise\Site 2\Packaging\Line 1\Holding Added to
Weinhardt's
```

system.recipe.cancelItemRecipe

Description

Cancel the current recipe for the production item specified by the itemPath parameter. If the production item is a line, then the recipe for all children production items of the line will also be cancelled.

Syntax

system.recipe. cancelItemRecipe(itemPath)

- Parameters

String itemPath - The item path to a production line, cell, cell group or location.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#---- cancelItemRecipe ----#
#-- function: system.recipe.cancelItemRecipe(itemPath)
#                                                (str)
# itemPath can be any line, cell, cell group or location

#-- required arguments
itemPath = '[global]\Enterprise\Site 2\Packaging\Line
1\Holding'

#-- execute
try:
    system.recipe.cancelItemRecipe(itemPath)
    print 'current recipe on ' + itemPath + ' Canceled'
except IOError:
    system.gui.messageBox('Request to cancel failed')
```

Output

```
current recipe on [global]\Enterprise\Site 2\Packaging\Line
1\Holding Canceled
```

system.recipe.changeRecipeGroup

Description

Change group of a specified recipe.

Syntax

system.recipe. changeRecipeGroup(recipeName, newRecipeGroup, note)

- Parameters

String recipeName - Name of the new recipe.

String newRecipeGroup - Name - this provides a way to filter out a subset of all recipes .

String note - Optional note to be stored in the recipe change log.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#---- changeRecipeGroup ----#
#-- function: system.recipe.changeRecipeGroup(recipeName,
newRecipeGroup, note)
#                                                (str)
(str)     (str)
#-- optional means that the field can be empty: "" or have a
value
```

```
#-- required rguments
recipeName = "Stubborn Soda"

#-- optional arguments
newRecipeGroup = "Contemporary"
note = 'Stubborn Soda group name changed from "" to
Contemporary on 3/20/17 by Nelson Kidd'

#-- execute
try:
    system.recipe.changeRecipeGroup(recipeName,
newRecipeGroup, note)
    print note
except IOError:
    system.gui.messageBox('Error - group name change failed')
```

Output

```
Stubborn Soda group name changed from "" to Contemporary on 3
/20/17 by Nelson Kidd
```

system.recipe.changeRecipeState

Description

Change state of a specified recipe.

Syntax

system.recipe. changeRecipeState (recipeName, newRecipeState, note)

- Parameters

String recipeName - Name of recipe.

String newRecipeState - Optional Name - this provides a way to filter out a subset of all recipes .

String note - Optional note to be stored in the recipe change log.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#---- changeRecipeState ----#
#-- function: system.recipe.changeRecipeGroup(recipeName,
newRecipeState, note)
#                                                (str)
(str)     (str)
#-- optional means that the field can be empty: "" or have a
value

#-- required arguments
recipeName = "IBC"

#-- optional arguments
newRecipeState = "Ready for testing"
note = 'IBC state changed from Hold to Ready for testing on 5/1
/17 by Robert Kellogg'

#-- execute
try:
    system.recipe.changeRecipeGroup(recipeName,
newRecipeGroup, note)
    print note
except IOError:
    system.gui.messageBox('Error - state change failed')
```

Output

```
IBC state changed from Hold to Ready for testing on 5/1/17 by
Robert Kellogg
```

system.recipe.createRecipe

system.recipe.createRecipe(recipeName, parentRecipeName, note)

Description

Create new recipe.

Syntax

system.recipe. createRecipe (recipeName, parentRecipeName, note)

- Parameters

String recipeName - Name of new recipe.

String parentRecipeName - Name of parent recipe on which to base this descendant recipe. Leave blank if new recipe is not based on any other recipe.

String note - Optional note to be stored in the recipe change log.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#---- createRecipe ----#
#-- function: system.recipe.createRecipe(recipeName,
parentRecipeName, note)
#                                          (str)
(str)       (str)
#-- optional means that the field can be empty: "" or have a
value
#-- required arguments
recipeName = "Bundaberg"
parentRecipeName = "RBC Master"
#-- optional arguments
note = 'Bundaberg added to RBC Master on 5/1/17 by Nolan Ryan'
#-- execute
try:
```

```
    system.recipe.createRecipe(recipeName, parentRecipeName,
note)
    print note
except IOError:
    system.gui.messageBox('Error - insert of Bundaberg failed')
```

```
Output

Bundaberg added to RBC Master on 5/1/17 by Nolan Ryan
```

system.recipe.createRecipe(recipeName, parentRecipeName, recipeState, recipeGroup, note)

Description

Create a new recipe.

Syntax

system.recipe. createRecipe (recipeName, parentRecipeName, recipeState, recipeGroup, note)

- Parameters

String recipeName - Name of new recipe.

String parentRecipeName - Name of parent recipe on which to base this descendant recipe. Leave blank if new recipe is not based on any other recipe.

String recipeState - Optional field you can you can filter by.

String recipeGroup - Optional field you can filter by.

String note - Optional note to be stored in the recipe change log.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#---- createRecipe ----#
#-- function: system.recipe.createRecipe(recipeName,
parentRecipeName, recipeState, recipeGroup, note)
#                                                (str)
(str)           (str)         (str)   (str)
#-- optional means that the field can be empty: "" or have a
value
#-- required arguments
recipeName = "Thomas Kemper"
parentRecipeName = "RBC Master"
#-- optional arguments
recipeState = "Hold"
recipeGroup = "New Blend"
note = 'Thomas Kemper added to RBC Master on Hold in the New
Blend group on on 5/1/17 by Carney Lansford'
#-- execute
try:
    system.recipe.createRecipe(recipeName, parentRecipeName,
note)
    print note
except IOError:
    system.gui.messageBox('Error - insert of Thomas Kemper
failed')
```

Output

```
Thomas Kemper added to RBC Master on Hold in the New Blend
group on on 5/1/17 by Carney Lansford
```

system.recipe.createSubProductCode

Description

Create a new sub product code (sub recipe).

Syntax

system.recipe. createSubProductCode(itemPath, subProductCode, note)

- Parameters

String itemPath - The item path to a production line, cell, cell group or location.

String subProductCode - New sub product code.

String note - Optional note to be stored in the recipe change log.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

Output

```
Remove this if it the snippet doesn't include print statements
```

system.recipe.deleteRecipe

Description

Deletes the specified recipe.

Syntax

system.recipe.deleteRecipe(recipeName, note)

- Parameters

String recipeName - Name of new recipe to delete.

String note - Optional note to be stored in the recipe change log.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#---- deleteRecipe ----#
#-- function: system.recipe.deleteRecipe(recipeName, note)
#                                          (str)   (str)
#-- optional means that the field can be empty: "" or have a
value

#-- required arguments
recipeName = "PC_007-IBC-RB"


#-- optional arguments
note = 'PC_007-IBC-RB deleted on 5/1/17 by Ozzie Smith'

#-- execute
try:
    system.recipe.deleteRecipe(recipeName, note)
    print note
except IOError:
    system.gui.messageBox('Error - delete of PC_007-IBC-RB
failed')
```

Output

```
PC_007-IBC-RB deleted on 5/1/17 by Ozzie Smith
```

system.recipe.deleteSubProductCode

Description

Delete sub product code.

Syntax

system.recipe. deleteSubProductCode (itemPath, subProductCode, note)

- Parameters

String itemPath - The item path to a production line, cell, cell group or location.

String subProductCode - Sub product code name.

String note - Optional note to be stored in the recipe change log.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

Output

```
Remove this if it the snippet doesn't include print statements
```

system.recipe.exportRecipe

Description

Adds a comment note to the current run for the selected line. See the Import / Export section of Editing Recipes for csv file format and other information.

Syntax

system.recipe. exportRecipe(filters)

- Parameters

String filters - Filter statements separated by commas. See the Recipe Analysis Provider for more information on the available filters.

- Returns

CSV formatted string containing the recipe values.

- Scope

All

Code Examples

Code Snippet

```
itemPath = event.source.parent.getComponent('MES Object
Selector').equipmentItemPath #Make sure the '[global]' project
is not in your Item Path.
filters = "Children=Include,Recipe Name=Master C,Item Path=%s"
% (itemPath)
csv = system.recipe.exportRecipe(filters)
system.file.writeFile("C:\\Temp\\recipe_export.csv", csv, False
)
```

Output

```
 Recipe_Name,Value_Name,Item_Path,Description,Units,Data_Type,
Format,Recipe_Value,Assigned_By
"PC_007-IBC-RB","Max Temperature","Enterprise\New
Site\Packaging\packagingLine1\Filler","","deg","Int4","#,##0.
##","92","Enterprise\New Site\Packaging\packagingLine1\Filler
- Default"
"PC_007-IBC-RB","Min Temperature","Enterprise\New
Site\Packaging\packagingLine1\Filler","","deg","Float8","#,##0.
##","85.3","Enterprise\New
Site\Packaging\packagingLine1\Filler - Default"
"PC_007-IBC-RB","IBC Sugar Percentage","Enterprise\New
Site\Packaging\packagingLine1\Filler","","","Float8","#,##0.
##","17.5","Enterprise\New
Site\Packaging\packagingLine1\Filler - Default"
"PC_007-IBC-RB","Line Speed","Enterprise\New
Site\Packaging\packagingLine1\Filler","","cpm","Int4","#,##0.
##","110","Enterprise\New Site\Packaging\packagingLine1\Filler
- Default"
```

system.recipe.getChangelogHistory

Description

Based on the filters set in the changelogFilters parameter, return change log history for recipe. See Recipe Change Log for more information.

Syntax

system.recipe.getChangelogHistory(changelogFilters)

- Parameters

ChangelogFilters changelogFilters - Change log filters (See ChangelogFilters object for more information).

- Returns

A Dataset object containing rows and columns of change log history.

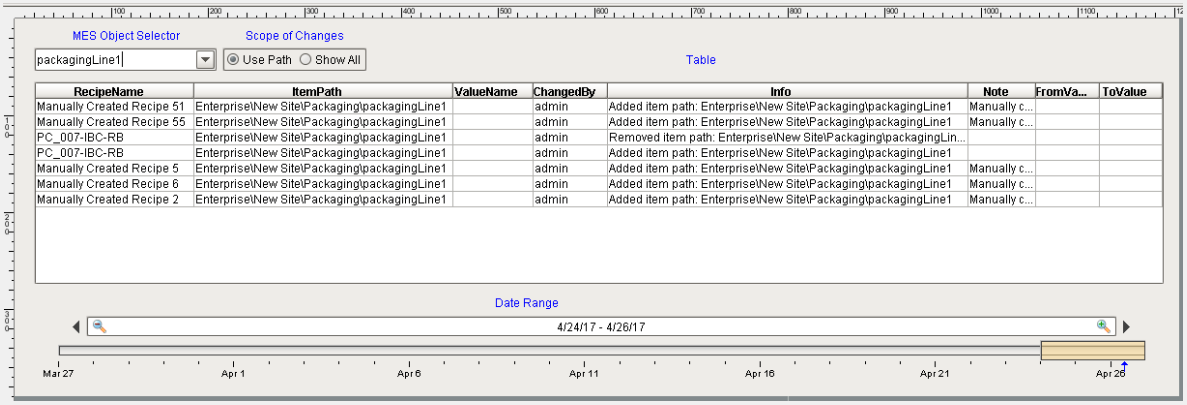
- Scope

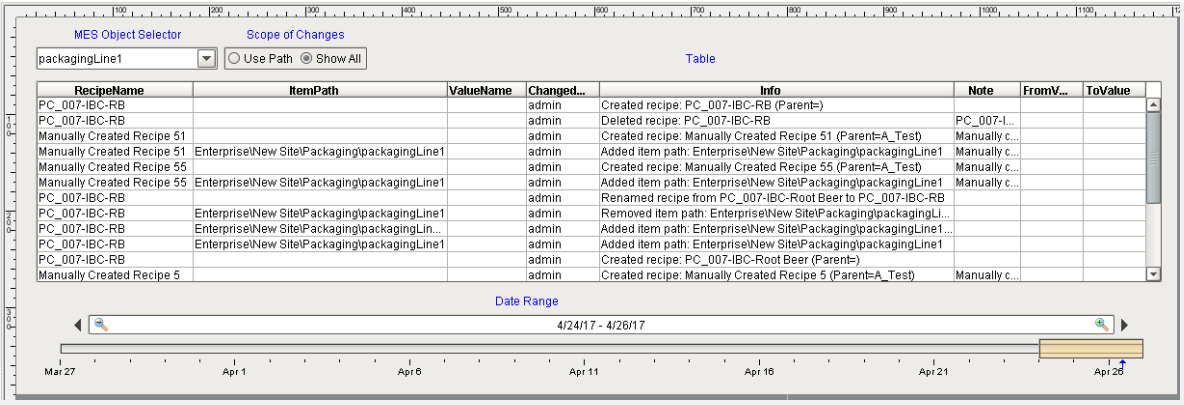
All

Code Examples

Code Snippet

```
#---- getChangelogHistory
#-- function: system.recipe.getChangelogHistory(filters)
#                                                  (str)
#-- see the screenshots of these test components

#-- find out if they want to see all changes or only those for
the selected itemPath
if event.source.parent.getComponent('cntPathorNot').
getComponent('rbUsePath').selected:
    itemPath = event.source.parent.getComponent('MES Object
Selector').equipmentItemPath
else:
    itemPath = ''

#-- Limit the data to a given date range
fromDate = event.source.parent.getComponent('Date Range').
startDate
toDate = event.source.parent.getComponent('Date Range').endDate

#-- Build the filters object
filters = system.recipe.filter.changelog.createNew()
filters.addCategory("Recipe")
filters.setItemPathFilter(itemPath)
filters.setFromDate(fromDate)
filters.setToDate(toDate)

#-- Request the change log for the given filters
ds = system.recipe.getChangelogHistory(filters)
event.source.parent.getComponent('Table').data = ds
```





system.recipe.getCurrentItemRecipe

Description

Return the current selected recipe name for a production item.

Syntax

system.recipe. getCurrentItemRecipe(itemPath)

- Parameters

String itemPath - The item path to a production line, cell, cell group or location.

- Returns

Current selected recipe name of specified production item.

- Scope

All

Code Examples

Code Snippet

```
#---- getCurrentItemRecipe----#
```

```
#-- function: system.recipe.getCurrentItemRecipe(itemPath)
#                                                  (str)
#---

itemPath = "[global]\Enterprise\New
Site\Packaging\packagingLine1\Filler"
currentRecipe = system.recipe.getCurrentItemRecipe(itemPath)
print 'Current Recipe = %s' %currentRecipe
```

Output

```
Current Recipe = PC_007-IBC-RB
```

system.recipe.getDefaultValues

Description

Return values for a sub recipe based on a product code or default values for a production item.

Syntax

system.recipe.getDefaultValues(itemPath, category, subProductCode)

- Parameters

String itemPath - The item path to a production line, cell, cell group or location.

String category - Category of recipe values to return. Where 1 is recipe values created by the recipe module, 2 is recipe values created by the OEE module and 3 is recipe values created by the SPC module. Use blank to include all categories.

String subProductCode - Sub product code to return values for, or else leave blank to read the default values for the production item.

- Returns

A list of Item Recipe Value objects.

- Scope

All

Code Examples

Code Snippet

```
#-- getDefaultValues
#-- function system.recipe.getDefaultValues(itemPath,
category, subProductCode)
#                                          (str)
(str)       (str)


#-- SCRIPT CONSOLE CODE
------------------------------------------------------------------
------------------------
#itemPath = 'Enterprise\Site
2\Packaging\packagingLine1\Filler' #change this to your path
#category = str('1')
#subProductCode = ''
#dv = system.recipe.getDefaultValues(itemPath, category,
subProductCode)
#if dv.size() > 0:
#    for value in dv:
#       print '%s = %s' %(value.getName(), value)
#else:
#   print 'No values created by the OEE module were Found'
#------------------------------------------------------------------
-----------------------------------------------


#-- WINDOW CODE
------------------------------------------------------------------
--------------------------------
#Put the following script in getDefaultValues button's
actionPerformed event handler
linePath = event.source.parent.getComponent('mesosLine').
equipmentItemPath
cell = event.source.parent.getComponent('mesosCell').
selectedName
itemPath = linePath + '\\' + cell
# the category selects from items created in various modules:
1=Recipe, 2=OEE, 3=SPC and ''=All
if event.source.parent.getComponent('cntSelections').
getComponent('rbAll').selected:
    category = str('')
    cat = 'All'
elif event.source.parent.getComponent('cntSelections').
getComponent('rbRecipe').selected:
    category = str('1')
```

```
    cat = 'Recipe'
elif event.source.parent.getComponent('cntSelections').
getComponent('rbSPC').selected:
    category = str('2')
    cat = 'SPC'
elif event.source.parent.getComponent('cntSelections').
getComponent('rbOEE').selected:
    category = str('3')
    cat = 'OEE'
# subProductCode  # leave blank to get values for the default
production item
if event.source.parent.getComponent('cntUseProductCode').
getComponent('rbUseProdCodeYes').selected:
    subProductCode = event.source.parent.getComponent('Product
Code Selector').selectedStringValue
else:
    subProductCode = ''

event.source.parent.getComponent('taData').text = ''
event.source.parent.getComponent('taData').text += '\n' + 'Choi
ces:'
event.source.parent.getComponent('taData').text += '\n' + 'Item
Path = %s' %(itemPath)
event.source.parent.getComponent('taData').text += '\n' + 'Cell
= %s' %(cell)
event.source.parent.getComponent('taData').text += '\n' + 'Cate
gory = %s' %(cat)
event.source.parent.getComponent('taData').text += '\n\n' + 'Va
lues:'

dv = system.recipe.getDefaultValues(itemPath, category,
subProductCode)
# dv returns an ArrayList - this list contains Item Recipe
Value objects

if dv.size() > 0:
    for value in dv:
        event.source.parent.getComponent('taData').text += '\n'
 + ' %s = %s' %(value.getName(), value)
        print '%s = %s' %(value.getName(), value)
else:
    event.source.parent.getComponent('taData').text += '\n' + '
No values created by the ' + cat + ' module were Found'

#-----------------------------------------------------------------
-------------------------------------------------
```

```
Line Speed(1) = 120
IBC Vanilla Percentage = 2.8
```

```
Max Temperature = 92
Min Temperature = 85.3
IBC Sugar Percentage = 17.5
Line Speed = 110
```

system.recipe.getItemLiveRecipeValues

Description

Returns a list of recipe value names and their current live values.

Syntax

system.recipe.getItemLiveRecipeValues (itemPath, recipeName, subProductCode, valueNames)

- Parameters

String itemPath - The item path to a production line, cell, cell group or location.

String recipeName - Name of the recipe to remove the specified production item.

String subProductCode - Sub product code to return values for, or else leave blank to read the default values for the production item.

String valueNames - The recipe value names to get the live values for, or leave blank for all recipe values.

- Returns

A list of recipe value names and their current live values.

- Scope

All

Code Examples

```
Code Snippet
```

```
# Get the current live values (tag values) for a recipe
#---- getItemLiveRecipeValues ----#
#-- function: system.recipe.getItemLiveRecipeValues(itemPath,
recipeName, subProductCode, valueNames)
#
(str)      (str)        (str)          (str)
#---

#--- SCRIPT CONSOLE CODE
------------------------------------------------------------------
------------
itemPath = "[global]\Enterprise\New
Site\Packaging\packagingLine1\Filler"
recipeName = "B_Test"
userRole = ''
result = ''
data = ''
map = system.recipe.getItemLiveRecipeValues(itemPath,
recipeName,"", "")
if map != None:
    for rv in map:
        data =  "%s\n   %s=%s" %(data, rv, map[rv])
    result = "%s\n\nLIVE values: %s" %(result, data)
    print result
else:
    print 'No live values found'
#------------------------------------------------------------------
-----------------------------------

#--- WINDOW CODE
------------------------------------------------------------------
-------------------
event.source.parent.getComponent('taOutput').text = ''
itemPath = event.source.parent.getComponent('mesosCell').
equipmentItemPath
#itemPath = "[global]\Enterprise\New
Site\Packaging\packagingLine1\Filler"
recipeName = event.source.parent.getComponent('RSC').
selectedRecipeName
#recipeName = "B_Test"
userRole = ''
result = ''
data = ''

# Get a map of the recipe value names and the current tag
values referenced by the recipe
map = system.recipe.getItemLiveRecipeValues(itemPath,
recipeName,"", "")
if map != None:
    for rv in map:
        data =  "%s\n   %s=%s" %(data, rv, map[rv])
```

```
    result = "%s\n\nLIVE values: %s" %(result, data)
    event.source.parent.getComponent('taOutput').text = result
else:
    event.source.parent.getComponent('taOutput').text = 'No
live values found'
#-------------------------------------------------------------
-----------------------------------
```

```
Output

LIVE values:
   IBC Vanilla Percentage=2.8
   Max Temperature=92
   Min Temperature=85.3
   IBC Sugar Percentage=17.5
   Line Speed=110
```

system.recipe.getItemRecipeList

getItemRecipeList(itemPath, recipeNameFilter)

Description

Return the current recipes available for a production item.

Syntax

system.recipe. getItemRecipeList(itemPath, recipeNameFilter)

- Parameters

String itemPath - The item path to a production line, cell, cell group or location.

String recipeNameFilter - Optional recipe filter. The filter can co ntain ? and * wild card characters.

- Returns

A list of currently available recipes.

- Scope

All

Code Examples

Code Snippet

```
#---- getItemRecipeList ? Returns a list of currently
available recipes ----#
#-- function: system.recipe.getItemRecipeList(itemPath, filter)
#                                              (str)     (str)
#-------------------------------------------------------------
--------------#
#--- Script Console Code
-------------------------------------------------------------
-------------------#
itemPath = "Enterprise\New
Site\Packaging\packagingLine1\Filler"
filter = "*e*"
list = system.recipe.getItemRecipeList(itemPath, filter)
if list.size() > 0:
    for recipeName in list:
        print 'Recipe Name = %s' %recipeName
else:
    print "No recipes meet the criteria of %s "  %filter
#-------------------------------------------------------------
-------------------------------------------#
#--- Window Code
-------------------------------------------------------------
--------------------------#
itemPath = event.source.parent.getComponent('mesosCell').
equipmentItemPath
filter = str(event.source.parent.getComponent('tfNameFilter 1')
.text)
event.source.parent.getComponent('taOutput').text = ''
list = system.recipe.getItemRecipeList(itemPath, filter)
if list.size() > 0:
    for recipeName in list:
        event.source.parent.getComponent('taOutput').text +=
recipeName + '\n'
else:
    event.source.parent.getComponent('taOutput').text += "No
recipes meet the criteria of " + filter
#-------------------------------------------------------------
-------------------------------------------#
```

Output

```
Recipe Name = B_Test
Recipe Name = Manually Created Recipe 1
Recipe Name = Manually Created Recipe 2
Recipe Name = Manually Created Recipe 3
Recipe Name = Manually Created Recipe 4
Recipe Name = Manually Created Recipe 5
Recipe Name = Manually Created Recipe 51
Recipe Name = Manually Created Recipe 55
Recipe Name = Manually Created Recipe 6
Recipe Name = ProcTest
```

getItemRecipeList(itemPath, recipeNameFilter, recipeStateFilter, recipeGroupFilter, includeMasterRecipes)

Syntax

getItemRecipeList(itemPath, recipeNameFilter, recipeStateFilter, recipeGroupFilter, includeMasterRecipes)

- Parameters

String itemPath - The item path to a production line, cell, cell group or location.

String recipeNameFilter - Optional recipe filter. The filter can co ntain ? and * wild card characters.

String recipeStateFilter - Optional recipe filter. The filter can contain ? and * wild card characters.

String recipeGroupFilter - Optional recipe filter. The filter can contain ? and * wild card characters.

boolean includeMasterRecipes - If true, returns also master recipes available for a production item.

- Returns

A list of currently available recipes.

- Scope

All

Code Examples

Code Snippet

```
#---- getItemRecipeList (with extra filters) ? returns a list
of currently available recipes based on the filter ----#
#-- function: system.recipe.getItemRecipeList(itemPath,
recipeNameFilter, recipeStateFilter, recipeGroupFilter,
includeMasterRecipes)
#                                                (str)
(str)            (str)              (str)
(bool)
#-- #-- optional means that the field can be empty: "" or have
a value
#-- required arguments
itemPath, includeMasterRecipes
#optional arguments
recipeNameFilter, recipeStateFilter, recipeGroupFilter
#--- Script Console Code
------------------------------------------------------------------
------------------------------------------------------
#--- getItemRecipeList with extra filters #
itemPath = "Enterprise\New
Site\Packaging\packagingLine1\Filler"
recipeNameFilter = "*e*"
recipeStateFilter = "*odd*"
recipeGroupFilter = "*pending*"
includeMasterRecipes = 1
list = system.recipe.getItemRecipeList(itemPath,
recipeNameFilter, recipeStateFilter, recipeGroupFilter,
includeMasterRecipes)
if list.size() > 0:
    for recipeName in list:
        print 'Recipe Name = %s' %recipeName
else:
    print "No recipes meet the criteria of the filters:"
    print "recipeNameFilter = %s "  %recipeNameFilter
    print "recipeStateFilter = %s "  %recipeStateFilter
    print "recipeGroupFilter = %s "  %recipeGroupFilter
    print "includeMasterRecipes = %s "  %includeMasterRecipes
#------------------------------------------------------------------
------------------------------------------------------------------
--------------
#--- Window Code
------------------------------------------------------------------
------------------------------------------------------------
#--- getItemRecipeList with extra filters #
includeMasterRecipes = event.source.parent.getComponent('cbIncl
udeMasterRecipes').selected
itemPath = event.source.parent.getComponent('mesosCell').
equipmentItemPath
```

```
#-- optional arguments
recipeNameFilter = str(event.source.parent.getComponent('tfName
Filter 2').text)
recipeStateFilter = str(event.source.parent.getComponent('tfSta
teFilter').text)
recipeGroupFilter = str(event.source.parent.getComponent('tfGro
upFilter').text)
event.source.parent.getComponent('taOutput').text = ''
list = system.recipe.getItemRecipeList(itemPath,
recipeNameFilter, recipeStateFilter, recipeGroupFilter,
includeMasterRecipes)
if list > 0:
    for recipeName in list:
        event.source.parent.getComponent('taOutput').text +=
recipeName + '\n'
else:
    event.source.parent.getComponent('taOutput').text += 'No
recipes meet the criteria of the filters: ' + '\n'
    event.source.parent.getComponent('taOutput').text += 'recip
eNameFilter: ' %recipeNameFilter + '\n'
    event.source.parent.getComponent('taOutput').text += 'recip
eStateFilter: ' %recipeStateFilter + '\n'
    event.source.parent.getComponent('taOutput').text += 'recip
eGroupFilter: ' %recipeGroupFilter + '\n'
    event.source.parent.getComponent('taOutput').text += 'inclu
deMasterRecipes: ' %includeMasterRecipes + '\n'
#-----------------------------------------------------------------
-----------------------------------------------------------------
--------------
```

Output

```
Recipe Name = Manually Created Recipe 5
Recipe Name = Manually Created Recipe 51
```

system.recipe.getProductionItemList

Description

Returns the list of production item corresponding to given recipe name and filter.

Syntax

system.recipe.getProductionItemList(recipeName, itemPathFilter)

- Parameters

String recipeName - The recipe name.

String itemPathFilter - The path to filter the results.

- Returns

A list of RecipeProductionItemInfo objects (See Recipe Production Item Info object in the MES documentation).

- Scope

All

Code Examples

Code Snippet

```
recipeName = 'PC_007-IBC-RB'
itemPathFilter = 'Enterprise\New Site\Packaging\packagingLine1'
list = system.recipe.getProductionItemList(recipeName,
itemPathFilter)
if list.size() > 0:
    print 'The production Items for %s:' %recipeName
    for productionItem in list:
        print productionItem
else:
    print 'No production items found for %s' %itemPathFilter
```

Output

```
The production Items for PC_007-IBC-RB:
packagingLine1
    CasePacker
    Checkweigher
    Filler
    Palletizer
    packingLine1 Quality
```

system.recipe.getRecipeValues

Description

Return recipe values for a production item and recipe combination.

Syntax

system.recipe.getRecipeValues(itemPath, recipeName, category)

- Parameters

String itemPath - The item path to a production line, cell, cell group or location.

String recipeName - Name of the recipe.

String category - Category of recipe values to return. Where '1' is recipe values created by the recipe module, '2' is recipe values created by the OEE module and '3' is recipe values created by the SPC module. Use blank to include all categories.

- Returns

A list of ItemRecipeValue objects (See Item Recipe Value object for more information).

- Scope

All

Code Examples

Code Snippet

```
recipeName = 'PC_007-IBC-RB'
itemPath = 'Enterprise\New
Site\Packaging\packagingLine1\Filler'
list = system.recipe.getRecipeValues(itemPath, recipeName,"")
if list.size() > 0:
    print 'The recipe values for %s:' %recipeName
    for rv in list:
        print "%s = %s" % (rv.getName(), str(rv.getValue()))
else:
    print 'No production items found for %s' %itemPathFilter
```

Output

```
The recipe values for PC_007-IBC-RB:
IBC Vanilla Percentage = 2.8
Max Temperature = 92
Min Temperature = 85.3
IBC Sugar Percentage = 17.5
Line Speed = 110
```

system.recipe.getRecipeValueSecurity

Description

Returns a Recipe Value Security Info object that contains each security roles settings. Retrieve a role by using the getSecurityRole method of the object.

Syntax

system.recipe.getRecipeValueSecurity(itemPath, valueName, inherited)

- Parameters

String itemPath - The item path to a production line, cell, cell group or location.

String valueName - The recipe value name to get the security settings for.

Boolean inherited - Set to 0 to get the value settings. If set to 1, it will return the inherited settings.

- Returns

A Recipe Value Security Info object.

- Scope

All

Code Examples

Code Snippet

```
# Get a recipe value security and display it

recipeName = "B_Test" #'PC_007-IBC-RB'
itemPath = 'Enterprise\New
Site\Packaging\packagingLine1\Filler'
userRole = 'Operators'
secdata = ""
result = ""

# first, get the list of all recipe value names and cycle
through the list
list = system.recipe.getRecipeValues(itemPath, recipeName,"")
if list.size() > 0:
    for rv in list:
        # get the security object (RecipeValueSecurityInfo)
for the recipe value name
        recipeValueSecurityInfo = system.recipe.
getRecipeValueSecurity(itemPath, rv.getName(),0)
        # get the RecipeValueSecurityRole object for a user
role from the security object
        recipeValueSecurityRole = recipeValueSecurityInfo.
getSecurityRoll(userRole)
        if recipeValueSecurityRole != None:
            secdata =  "%s \n %s: userRole=%s min val=%s, max
val=%s" % (secdata,rv.getName(), userRole, str(recipeValueSecur
ityRole.getMinValue()), str(recipeValueSecurityRole.
getMaxValue()) )
    result = "%s\n\nSecurity values: %s" %(result, secdata)
    print result
else:
    print 'No production items found for %s' %itemPathFilter
```

Output

```
Security values:
 IBC Vanilla Percentage: userRole=Operators min val=-
1.7976931348623157E308, max val=1.7976931348623157E308
 Max Temperature: userRole=Operators min val=-2.147483648E9,
max val=2.147483647E9
 Min Temperature: userRole=Operators min val=-
1.7976931348623157E308, max val=1.7976931348623157E308
 IBC Sugar Percentage: userRole=Operators min val=-
1.7976931348623157E308, max val=1.7976931348623157E308
 Line Speed: userRole=Operators min val=-2.147483648E9, max
val=2.147483647E9
```

system.recipe.getRecipeVariances

Description

Based on the filters set in the varianceFilters parameter, return recipe value variances. See Variance Monitoring for more information.

Syntax

system.recipe.getRecipeVariances(varianceFilters)

- Parameters

Variance Filters varianceFilters - Change log filters (See Variance Filters object for more information).

- Returns

A Dataset object containing rows and columns of recipe value variances.

- Scope

All

Code Examples

Code Snippet

```
#Collection values we want to filter by
projectName = system.util.getProjectName()
itemPath = event.source.parent.getComponent('Production Line
Selector').selectedPathWithoutProject

#Build the filters object
filters = system.recipe.filter.variance.createNew()
filters.setProjectName(projectName)
filters.setVarianceEntryType("Recipe")
filters.setVarianceScopeTypes("Last")
filters.setItemPath(itemPath)
```

```
#Request the variances for the given filters
ds = system.recipe.getRecipeVariances(filters)
event.source.parent.getComponent('Table').data = ds
```

Output

system.recipe.importRecipe

Description

Set the recipe values to the current tag value(s) for the production item specified by the itemPath parameter. See the Import / Export section of Editing Recipes for CSV file format and other information.

Info

Values that are outside of the range defined in the recipe values security will not be imported. When this happens, an exception is returned listing all of the values that were not imported. See Recipe Security for more information.

Syntax

system.recipe. importRecipe(csvData, note)

- Parameters

String csvData - String in CSV format containing recipe values. Must be the same format that is returned by the exportRecipe function.

String note - Optional note to be stored in the recipe change log.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
csv = system.file.readFileAsString("C:\\Temp\\recipe_export.
csv")
system.recipe.importRecipe(csv, "Values set during import.")
```

Output

system.recipe.isItemRecipeMonitoringEnabled

Description

Return recipe value monitoring enabled state. See Variance Monitoring for more information.

Syntax

system.recipe. isItemRecipeMonitoringEnabled(itemPath)

- Parameters

String itemPath - The item path to a production line, cell, cell group or location.

- Returns

True, if recipe value monitoring is enabled.

- Scope

All

Code Examples

```
Code Snippet
```

```
Output

Remove this if it the snippet doesn't include print statements
```

system.recipe.readItemCurrentValues

Description

Set the recipe values to the current tag value(s) for the production item specified by the itemPath parameter.

Syntax

system.recipe.readItemCurrentValues(itemPath, includeChildren, recipeName, subProductCode, valueNames, note)

- Parameters

String itemPath - The item path to a production line, cell, cell group or location.

Boolean includeChildren - If true, also set the recipe values for children of the items.

String recipeName - Name of recipe.

String subProductCode - The sub product code.

String valueNames - The name(s) of the recipe values to set to current tag values. Separate multiple recipe value names with commas. For all recipe values of a production item, leave blank.

String note - Optional note to be stored in the recipe change log.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet
```

```
Output

Remove this if it the snippet doesn't include print statements
```

system.recipe.removeItemFromRecipe

Description

Remove a production item from a recipe.

Syntax

system.recipe. removeItemFromRecipe(recipeName, itemPath, note)

- Parameters

String recipeName - Name of new recipe to remove the specified production item.

String itemPath - The item path to the production line, cell, cell group or location to remove from the recipe.

String note - Optional note to be stored in the recipe change log.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet
```

```
Output

Remove this if it the snippet doesn't include print statements
```

system.recipe.renameRecipe

Description

Rename specified recipe.

Syntax

system.recipe. renameRecipe (existingRecipeName, newRecipeName, note)

- Parameters

String existingRecipeName - Name of the existing recipe.

String newRecipeName - New recipe name.

String note - Optional note to be stored in the recipe change log.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet
```

```
Output

Remove this if it the snippet doesn't include print statements
```

system.recipe.renameSubProductCode

Description

Adds a comment note to the current run for the selected line.

Syntax

system.recipe. renameSubProductCode (itemPath, existingSubProductCode, newSubProductCode, note)

- Parameters

String linePath - The item path to a production line, cell, cell group or location.

String existingSubProductCode - Existing sub product code name.

String newSubProductCode - New sub product code name.

String note - Optional note to be stored in the recipe change log.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet
```

```
Output

Remove this if it the snippet doesn't include print statements
```

system.recipe.revertPathDefaultValue

Description

Revert production item default values back to be inherited from the parent.

Syntax

system.recipe.revertPathDefaultValue(itemPath, subProductCode, valueNames, note)

- Parameters

String itemPath - The item path to a production line, cell, cell group or location.

String subProductCode - Sub product code to set value for, or else leave blank to set the default value for the production item.

String valueNames - One or more recipe value names separated by commas to revert.

String note - Optional note to be stored in the recipe change log.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet
```

```
Output

Remove this if it the snippet doesn't include print statements
```

system.recipe.revertPathRecipeValues

Description

Revert production item recipe values back to the parent production item.

Syntax

system.recipe. revertPathRecipeValues (itemPath, recipeName, valueNames, note)

- Parameters

String itemPath - The item path to a production line, cell, cell group or location.

String recipeName - Name of recipe.

String valueNames - One or more recipe value names separated by commas to revert.

String note - Optional note to be stored in the recipe change log.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet
```

```
Output

Remove this if it the snippet doesn't include print statements
```

system.recipe.setItemRecipe

Description

Set the recipe for the production item specified by the itemPath parameter. If the production item is a line, then all children production items of the line will also be set to the same recipe provided they were added to the recipe.

Syntax

system.recipe. setItemRecipe(itemPath, recipeName, enableValueMonitoring)

- Parameters

String itemPath - The item path to a production line, cell, cell group or location.

String recipeName - Name of recipe.

Boolean enableValueMonitoring - If true, turn on recipe value variance monitoring. See Variance Monitoring for more information.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
#---- setItemRecipe ----#
#-- function: system.recipe.setItemRecipe(itemPath,
recipeName,  enableValueMonitoring)
#                                          (str)
(str)              (bool)
#-- Definitions
itemPath = '[global]\Enterprise\Site 2\Packaging\Line
1\Holding'
recipeName = 'Stubborn'
enableValueMonitoring = True
if enableValueMonitoring == True:
    evm = 'On'
else:
    evm = 'Off'
#-- execute
try:
    system.recipe.setItemRecipe(itemPath, recipeName,
enableValueMonitoring)
     print recipeName + ' selected for ' + itemPath + ' and
value monitoring is ' + evm
except:
    system.gui.messageBox('Attempt to set recipe item failed')
```

Output

```
Stubborn selected for [global]\Enterprise\Site
2\Packaging\Line 1\Holding and value monitoring is On
```

system.recipe.setPathDefaultValue

Description

Set a production item sub recipe or default value.

Syntax

system.recipe. setPathDefaultValue (itemPath, subProductCode, valueName, value, note)

- Parameters

String itemPath - The item path to a production line, cell, cell group or location.

String subProductCode - Sub product code to set value for, or else leave blank to set the default value for the production item.

String valueName - Recipe value name to set the value.

String value - Set the recipe value to this value.

String note - Optional note to be stored in the recipe change log.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet
```

```
Output
```

```
Remove this if it the snippet doesn't include print statements
```

system.recipe.setPathRecipeValue

Description

Set production item recipe value.

Syntax

system.recipe. setPathRecipeValue(itemPath, recipeName, valueName, value, note)

- Parameters

String itemPath - The item path to a production line, cell, cell group or location.

String recipeName - Name of recipe.

String valueName - Recipe value name to set.

String value - New value to assign to recipe value.

String note - Optional note to be stored in the recipe change log.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet

#--- setPathRecipeValue(itemPath, recipeName, valueName,
value, note) ---
#                         (str)       (str)      (str)    (str)
(str)
```

```
#
# function system.recipe.setPathRecipeValue() allows you to
set the value defined on
# a production Item. You can verify the change using the
Recipe Editor. After you do
# a setItemRecipe() the associated tag will be update to this
value

#-- Window Code
itemPath = event.source.parent.getComponent('mesosCell').
equipmentItemPath
recipe = event.source.parent.getComponent('selRecipeName').
selectedRecipeName
valueName = event.source.parent.getComponent('cntPrv').
getComponent('tfPrvValueName').text
value = event.source.parent.getComponent('cntPrv').getComponent
('tfPrvValue').text
note = event.source.parent.getComponent('cntPrv').getComponent(
'tfPrvNote').text
if valueName == '' or value == '':
    event.source.parent.getComponent('taOutput').text = 'Please
fill in all required fields!'
else:
    try:
        event.source.parent.getComponent('taOutput').text = ''
        event.source.parent.getComponent('taOutput').text += 'i
temPath: %s' %itemPath + '\n'
        event.source.parent.getComponent('taOutput').text += 'v
alueName: %s' %valueName + '\n'
        event.source.parent.getComponent('taOutput').text += 'v
alue: %s' %value + '\n'
        event.source.parent.getComponent('taOutput').text += 'n
ote: %s' %note + '\n'
        system.recipe.setPathRecipeValue(itemPath, recipe,
valueName, value, note)
    except IOError:
        event.source.parent.getComponent('taOutput').text = 'At
tempt to set the recipe value failed!'

#-- Script Console Code
# itemPath = "Enterprise\New
Site\Packaging\packagingLine1\Filler"
# recipe = "Stubborn"
# valueName = "IBC Sugar Percentage"
# value = "17.4"
# note = "17.4 IBC Sugar"
# try:
#     system.recipe.setPathRecipeValue(itemPath, recipe,
valueName, value, note)
#     system.gui.messageBox('Stubborn IBC Sugar Percentage
updated to 17.4', 'setPathRecipeValue')
```

```
# except IOError:
#     system.gui.messageBox('Attempt to set the recipe value
failed!', 'setPathRecipeValue')
```

Output

```
itemPath: Enterprise\New Site\Packaging\packagingLine1\Filler
valueName: IBC Sugar Percentage
value: 17.7
note: Note for IBC Sugar
```

system.recipe.setPathRecipeValues

Description

Set multiple recipe values for a production item. The recipe must be the same for all recipe values being set.

Syntax

system.recipe. setPathRecipeValues (itemPath, recipeName, values, note)

- Parameters

String itemPath - The item path to a production line, cell, cell group or location.

String recipeName - Name of recipe.

PyDictionary values - A Python Dictionary containing recipe values in a name / value format.

String note - Optional note to be stored in the recipe change log.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet
```

```
Output

Remove this if it the snippet doesn't include print statements
```

system.recipe.updateRecipeValueSecurity

Description

Updates the security settings for a recipe.

Syntax

system.recipe.updateRecipeValueSecurity(securityInfo)

- Parameters

Recipe Value Security Info securityInfo - The security setting to be updated.

- Returns

The updated recipe setting.

- Scope

All

Code Example 1

Code Snippet

```
secInfo = system.recipe.getRecipeValueSecurity('Enterprise\Site
\Area\Recipe Test 1\Recipe Test 1A', 'RV 1', True)

#Cycle through and print the setting for each roll
for ndx in range(secInfo.getSecurityRollCount()):
    recSec = secInfo.getSecurityRoll(ndx)
    print recSec.getSecurityRoll()
    print recSec.isAllowEdit()
    print recSec.getMinValue()
    print recSec.getMaxValue()
    print

#Get a security roll for by name
secRoll = secInfo.getSecurityRoll('Recipe Test')
print secRoll.getSecurityRoll()
print secRoll.isAllowEdit()
print secRoll.getMinValue()
print secRoll.getMaxValue()
print

#Change the settings of the security roll
secRoll.setAllowEdit(True)
secRoll.setMinValue(10.0)
secRoll.setMaxValue(100.0)

#This must be set otherwise it will inherit from the parent
secInfo.setInherit(False)

#Update the security settings
system.recipe.updateRecipeValueSecurity(secInfo)
```

Output

Code Example 2

Code Snippet

```
#This will revert to inherit the security from the parent
#Get security information for a recipe value
```

```
secInfo = system.recipe.getRecipeValueSecurity('Enterprise\Site
\Area\Recipe Test 1\Recipe Test 1A', 'RV 1', True)

#Reset the settings to inherit from the parent
secInfo.setInherit(True)

#Update the security settings
system.recipe.updateRecipeValueSecurity(secInfo)
```

Output

9.7.10 system.instrument

system.instrument.parse.parseText

Description

Parse string data contained in the "text" property using the template specified by the "templateName" property.

Info

The function requires an additional argument on the Gateway script which is the name of the project.

Syntax

system.instrument.parse.parseText(templateName, text)

- Parameters

String templateName - The name of the parse template to use to parse the text.

String text - The text to parse.

- Returns

ParseResults - Returns a ParseResults object containing the parsed values. See ParseResults object reference for more information on how to get values from the results.

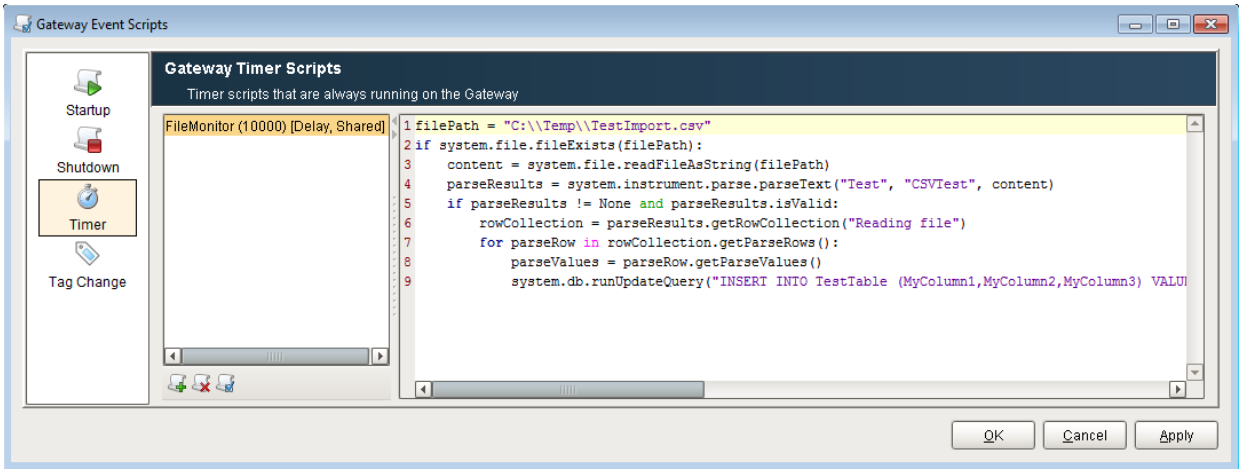
- Scope

All

Code Examples

Code Snippet

```
#Sample script to read and parse a CSV file then convert the
parse results to a dataset and display in a table component:
fileStr = system.file.readFileAsString("C:\\Temp\\Test.csv")
parseResults = system.instrument.parse.parseText("CSV Test
Column", fileStr)
if parseResults.isValid():
 dataset = parseResults.createDataset("CSV Results")
 event.source.parent.getComponent('Table').data = dataset
```



system.instrument.parse.types.valueOf

Description

Syntax

system.instrument.parse.types.valueOf()

- Parameters

Type name - description

- Returns

Type - description

- Scope

All

Code Examples

Code Snippet

Output

```
Remove this if it the snippet doesn't include print statements
```

system.instrument.parse.types.values

Description

Syntax

system.instrument.parse.types.values()

- Parameters

Type name - description

- Returns

Type - description

- Scope

All

Code Examples

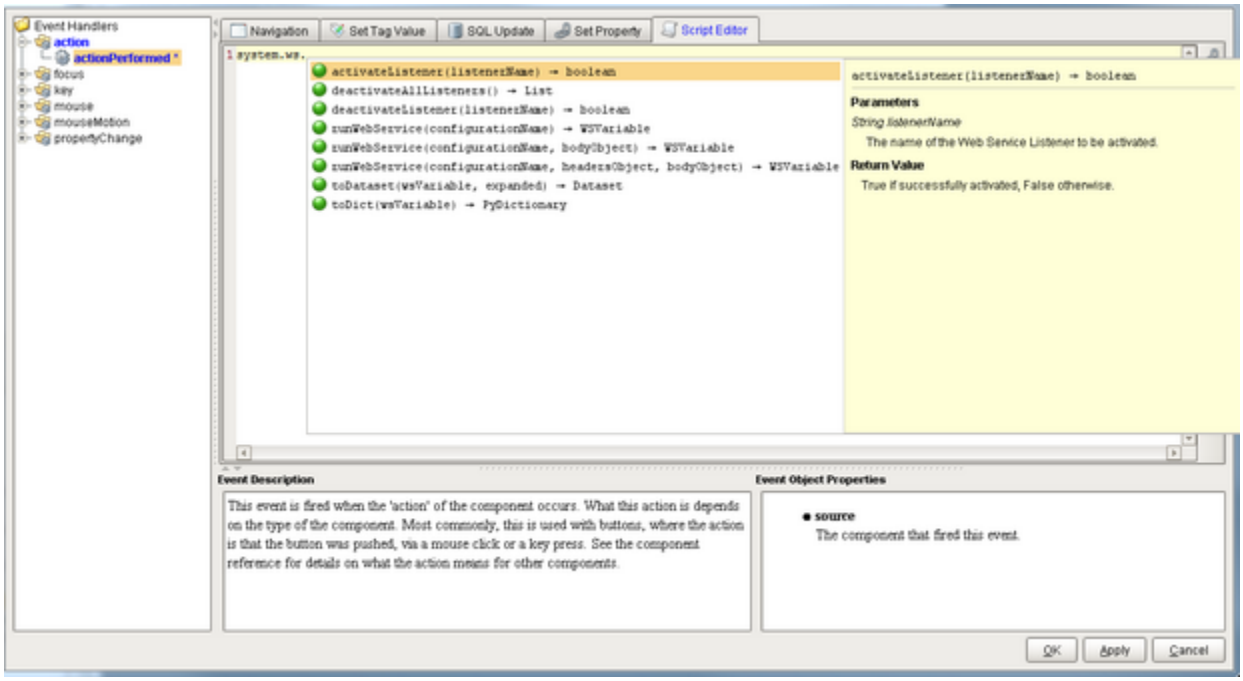
```
Code Snippet
```

```
Output

Remove this if it the snippet doesn't include print statements
```

9.7.11 system.ws

The Web Service Module exposes many script functions that support associated functions. These script functions are described in system.ws.

In the Ignition script editor, the documentation for the script functions can be accessed by pressing control-space after typing in "system.". For all the Web Service script functions, type in " system.ws ." and press control-space to see the associated function and documentation.



WS_ScriptIntro

Ignition Script Auto Document Feature

system.ws.runWebService

system.ws.runWebService(configurationName)

Description

This will run a web service with the current parameter settings that are defined in the configuration.

Syntax

system.ws **.runWebService(configurationName)**

- Parameters

String configurationName - The name of the Web Service Configuration to run.

- Returns

A Web Service Variable object that represents the result of the Web Service.

- Scope

All

Code Examples

Code Snippet

```
result = system.ws.runWebService("LengthConvertor")
print result
```

Output

```
{"Root": {"ChangeLengthUnitResponse":
{"ChangeLengthUnitResult": "0.83333333333333348"}}}
```

system.ws.runWebService(configurationName, bodyObject)

Description

This will run a web service with the current parameter settings that are defined in the configuration.

Syntax

system.ws **.runWebService(configurationName , bodyObject)**

- Parameters

String configurationName - The name of the Web Service Configuration to run.

PyDictionary bodyObject - This will replace any parameters already defined by the configuration with a python object that will be used as the parameters of the web service. Using a python dictionary is recommended.

- Returns

A Web Service Variable object that represents the result of the Web Service.

- Scope

All

Code Examples

```
Code Snippet

miles = event.source.parent.getComponent('Numeric Text Field
Miles').doubleValue
result = system.ws.runWebService('Measurement Conversion', {'Le
ngthValue' : miles})
kilometers = result.getChild('ChangeLengthUnitResult')
event.source.parent.getComponent('Numeric Label Kilometers').
text = str(kilometers.getValue())
```

The first line gets the miles value from the Numeric Text Field component.

The second line calls the web service configuration and passes the miles value parameter. The fromLengthUnit and toLengthUnit parameters don't have to be passed here because they were defined in step 4. However, they can be passed here and it will override the values in the Measurement Conversion configuration. The body object parameter is a python object.

Line 3 read the kilometers value from the results. Because multiple values might exist in the results, this is done by name.

Line 4 converts the kilometer value to a string and puts it into the Numeric Label component.

system.ws.runWebService(configurationName, headersObject, bodyObject)

Description

This will run a web service with the current parameter settings that are defined in the configuration.

Syntax

system.ws **.runWebService(configurationName , headersObject, bodyObject)**

- Parameters

String configurationName - The name of the Web Service Configuration to run.

PyDictionary headersObject - This will replace and headers already defined by the configuration with a python object that will be used as the headers of the web service. Using a python dictionary is recommended.

PyDictionary bodyObject - This will replace any parameters already defined by the configuration with a python object that will be used as the parameters of the web service. Using a python dictionary is recommended.

- Returns

A Web Service Variable object that represents the result of the Web Service.

- Scope

All

Code Examples

Code Snippet

```
result = system.ws.runWebService("Post_Article", {"Cache-
Control" : "no-cache"}, {"title" : "This is a title.", "text"
: "This is a text."})
print result
```

Output

```
{"Root": {
  "id": 1,
  "title": "This is a title.",
  "text": "This is a text."
}}
```

system.ws.runWebService(configurationName, urlParam, headersObject, bodyObject)

Description

This will run a web service with the current parameter settings that are defined in the configuration.

Syntax

system.ws **.runWebService(configurationName , urlParams, headersObject, bodyObject)**

- Parameters

String configurationName - The name of the Web Service Configuration to run.

PyDictionary urlParam- This will replace the URL resource path or URL query string already defined by the configuration with a python object that will be used as the URL resource path or URL query string of the web service. Using a python dictionary is recommended.

PyDictionary headersObject - This will replace and headers already defined by the configuration with a python object that will be used as the headers of the web service. Using a python dictionary is recommended.

PyDictionary bodyObject - This will replace any parameters already defined by the configuration with a python object that will be used as the parameters of the web service. Using a python dictionary is recommended.

- Returns

A Web Service Variable object that represents the result of the Web Service.

- Scope

All

Code Examples

Code Snippet

```
result = system.ws.runWebService("Put_Article", {"id" : 1}, None, {"title" : "This is a title.", "text" : "This is a text."})
print result
```

Output

```
{"Root": {
  "id": 1,
  "title": "This is a title.",
  "text": "This is a text."
}}
```

system.ws.setDefaultMaxConnectionPerRoute

Description

Sets the default maximum number of connections allowed per route.

Syntax

system.ws.setDefaultMaxConnectionPerRoute(max)

- Parameters

int max - The default maximum number of connections allowed per route.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
system.ws.setDefaultMaxConnectionPerRoute(100)
```

system.ws.setIdleConnectionTimeout

Description

Sets the timeout to close idle connections.

Syntax

system.ws **.setIdleConnectionTimeout(idleTimeout)**

- Parameters

long idleTimeout - The idle connection time in seconds.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
system.ws.setIdleConnectionTimeout(100000)
```

system.ws.setMaxConnectionPerRoute

Description

Sets the maximum number of connections allowed per route.

Syntax

system.ws.setMaxConnectionPerRoute(route, max)

- Parameters

String route - The connection route.

int max- The maximum number of connections allowed of the connection route.

- Returns

Nothing

- Scope

All

Code Examples

Code Snippet

```
system.ws.setMaxConnectionPerRoute("http://route1.test.com", 20
)
```

system.ws.setMaxTotalConnection

Description

Sets the maximum number of total connections allowed.

Syntax

system.ws.setMaxTotalConnection(max)

- Parameters

int max - The maximum number of total connections allowed.

- Returns

Nothing

- Scope

All

Code Examples

```
Code Snippet

system.ws.setMaxTotalConnection(400)
```

system.ws.toDataset

Description

Converts a web service variable into a dataset.

Syntax

system.ws **.toDataset(wsVariable, expanded)**

- Parameters

WSVariable wsVariable - The web service variable to be converted. Must be complex.

Boolean expanded - If true, the returned dataset and all contained datasets will be expanded to better represent a dataset structure. If false, it will directly represent the WSDL defined schema.

- Returns

A dataset representing the given Web Service Variable.

- Scope

All

Code Examples

Code Snippet

```
result = system.ws.runWebService("LengthConvertor")
print system.ws.toDataset(result, True)
```

system.ws.toDict

Description

Converts a web service variable into a python dictionary.

Syntax

system.ws **.toDict(wsVariable)**

- Parameters

WSVariable wsVariable - The web service variable to be converted. Must be complex.

- Returns

A python dictionary representing the given Web Service Variable.

- Scope

All

Code Examples

Code Snippet

```
result = system.ws.runWebService("LengthConvertor")
print system.ws.toDict(result)
```

```
Output

{u'Root': {u'ChangeLengthUnitResponse':
{u'ChangeLengthUnitResult': u'0.083333333333333343'}}}
```

9.7.12 system.barcode.scanner

system.barcode.scanner.decode

Description

Returns a Decode Results object with the results of the decoding process.

Syntax

system.barcode.scanner.decode(rawBarcode, method, patterns, preamble, postamble, separator)

- Parameters

String rawBarcode - Raw barcode data to decode.

String method - Use “SinglePass” to do a single pass search of each pattern in the patterns list, or use “Consume” to do a multi-pass search where each pattern in the list is search for at the beginning of the raw barcode string, then remove from the string until the raw barcode is consumed. A “Consume” process can result in an unmatched portion of the raw barcode left and placed in the DecodeResults.getUnmatched property. When decoding a GS1 standard barcode the “Consume” method should be used.

String patterns - A list of barcode patterns to be used to search the raw barcode.

String preamble - The prefix regex string to be removed from the raw barcode prior to doing the pattern matching. A Unicode character of \u0002 for STX can be used.

String postamble - The suffix regex string to be removed from the raw barcode prior to doing the pattern matching. A Unicode character of \u0003 for ETX or \u000A for new line /line feed or \u000D for carriage return can be used.

String separator - The GS1 FNC1 separator as a regex string used if decoding a GS1 barcode. A Unicode character of \u001D or \u00E8 can be used.

- Returns

A Decode Results object.

- Scope

All

Code Examples

Code Snippet

```
```

Output

```
Remove this if it the snippet doesn't include print statements
```

system.barcode.scanner.getNewBarcodePattern

Description

Returns a Barcode Pattern object to be used to append to a List<BarcodePattern>

Syntax

system.barcode.scanner.getNewBarcodePattern(name, key, pattern)

- Parameters

String name - Name for the pattern.

String key - Key for the pattern. This key is used to refer the resultant hash table or Python dictionary object.

String pattern - Regular expression pattern.

- Returns

A Barcode Pattern object.

- Scope

All

Code Examples

```
Code Snippet
```

```
Output

Remove this if it the snippet doesn't include print statements
```

system.barcode.scanner.getPatternList

Description

Returns a subset list of the predefined barcode patterns that can be used to decode a barcode.

Syntax

system.barcode.scanner.getPatternList(fillKeys, separator)

- Parameters

String fillKeys - A list of keys to the predefined list of patterns to load into the returned list. Use “All” to load all predefined patterns. Or for example use “GS1-10,GS1-17, GS1-310” to just get the 3 GS1 AI patterns 10, 17, 310 respectively.

String separator - An optional separator to be used for patterns that can be of variable length and need a separator code. For GS1 FNC1 a unicode character of \u001D or \u00E8 can be used.

- Returns

A list of barcode patterns.

- Scope

All

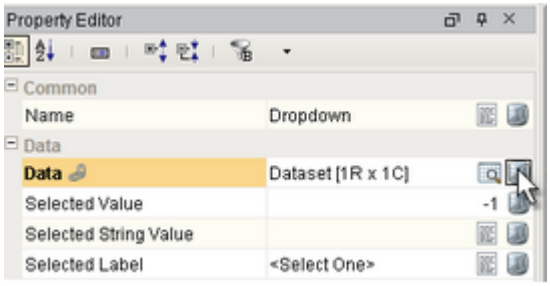
Code Examples

```
Code Snippet
```

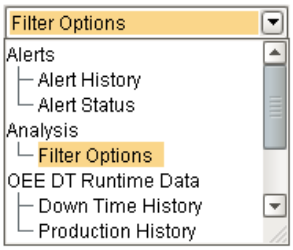
```
Output

Remove this if it the snippet doesn't include print statements
```

9.8 Binding Functions

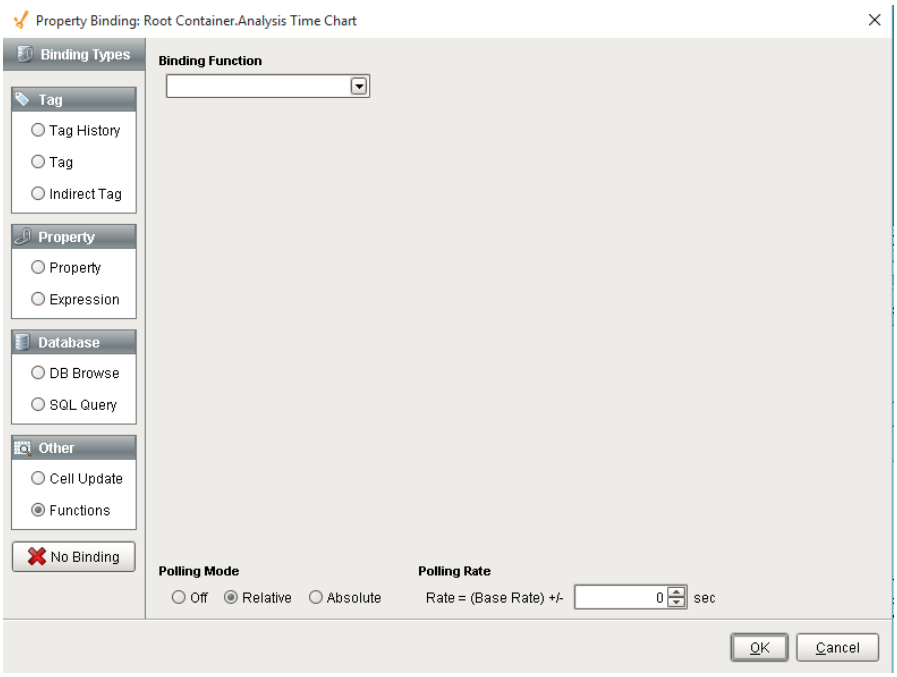
The MES Modules provide a set of 'Function' bindings that you can use with components to return datasets. The function bindings will bring back information from the analysis engine. To access the binding functions, click on the icon of a component property as shown below.



The binding options window will appear. Next click on the Functions option and select one of the binding functions from the drop-down list.



The parameters that are associated with the selected binding function will appear. Each of these parameters can accept a constant value, bound to a property of another component, or bound to a SQLTag.

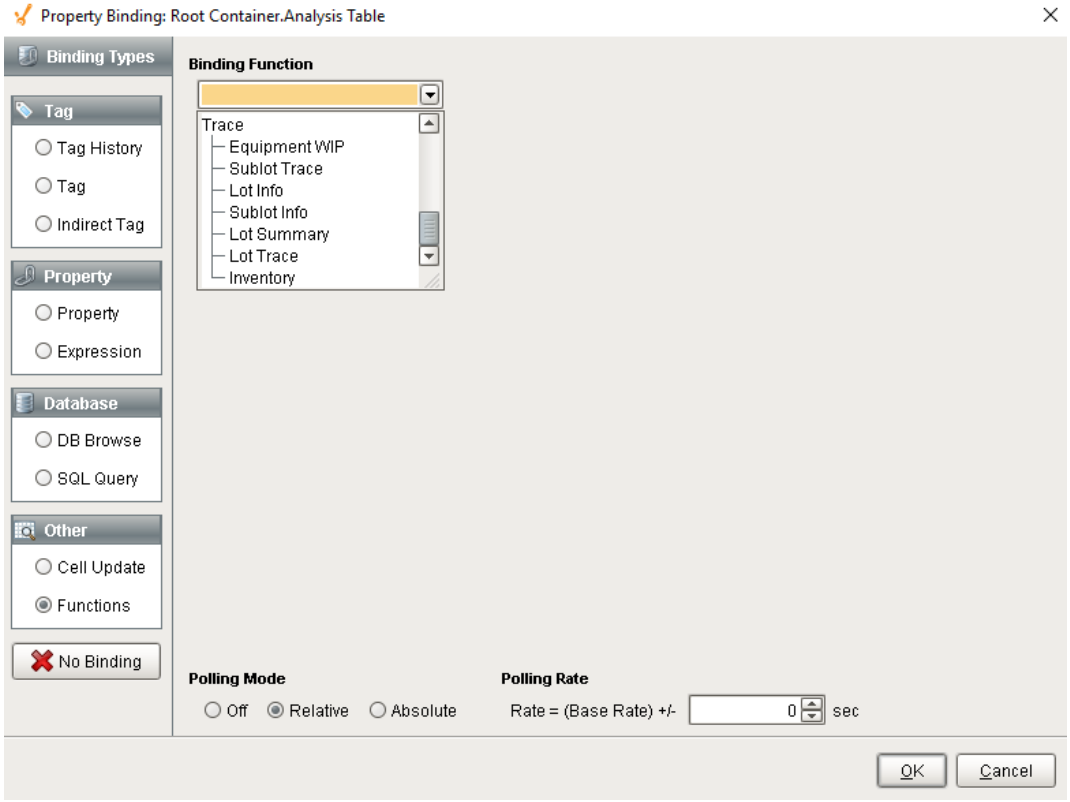


Once the parameters have been set and the polling mode selected, the server will return the results based on the provided parameter values.

9.8.1 Trace

Besides using the track and trace components or scripting, the binding functions can be used to retrieve trace information that can be used in reports or non track and trace components.

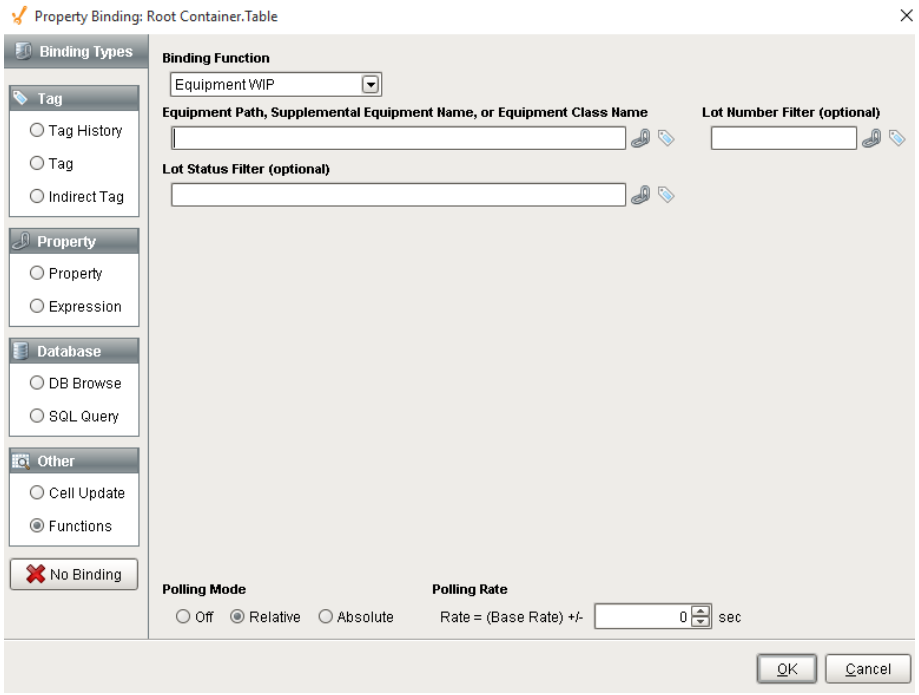
The image below shows the MES property binding screen that is commonly used when using building in Ignition projects. To access the track and trace binding functions, select Functions and then select the desired binding function in the Trace group.



Equipment WIP

Description

The Equipment WIP binding function is used to retrieve information for a lot based on the filter parameters listed in the image below. It can be used to display and report lot details including Equipment Path, Supplemental Equipment Name and Equipment Class Name.



Function Name

Filter Options

Parameters

Name	Description	Property Type
Equipment Path, Supplemental Equipment Name, or Equipment Class Name.	The Equipment Path, Supplemental Equipment Name, or Equipment class name used to filter the results.	String
Lot Number Filter	This is optional. Custom lot number to filter results. Filter value, including * and ? wildcard characters, to filter results by lot number.	String
Lot Status Filter	This is optional. Custom lot status value to filter results. Filter value, including * and ? wildcard characters, to filter results by lot status.	String

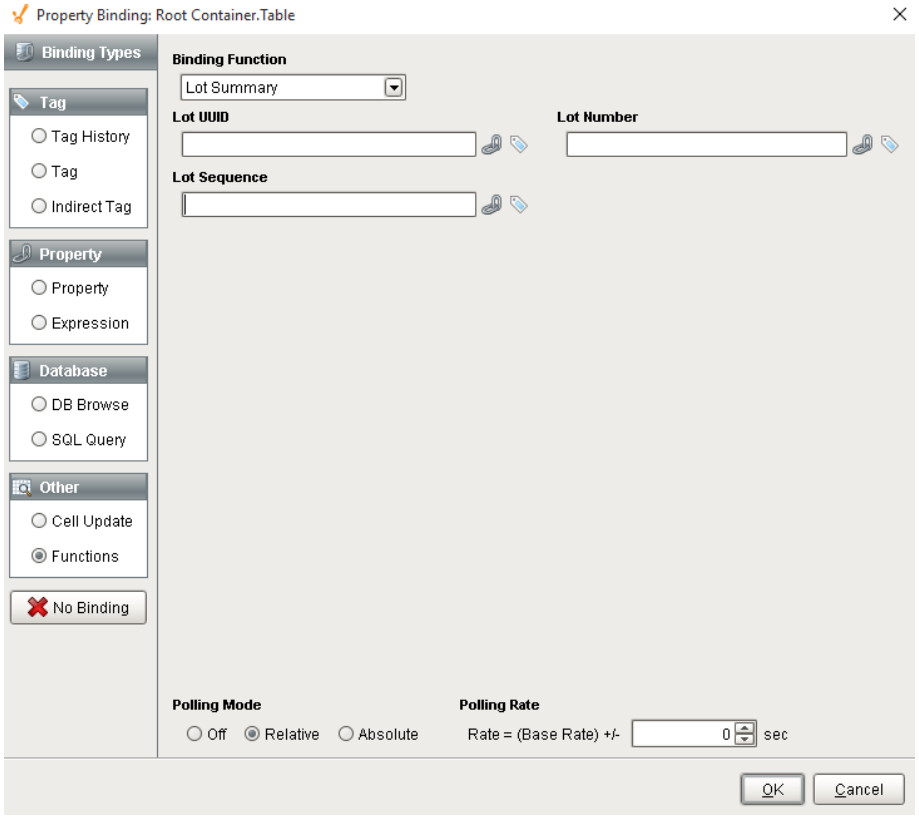
Info

For more information on scripting, see system.mes.getLotInventoryByEquipment.

Lot Summary

Description

The Lot Summary binding function is used to retrieve information for a lot based on the filter parameters listed in the image below. It can be used to display and report lot details including Lot UUID, Lot Number and Lot Sequence.



Function Name

Lot Summary

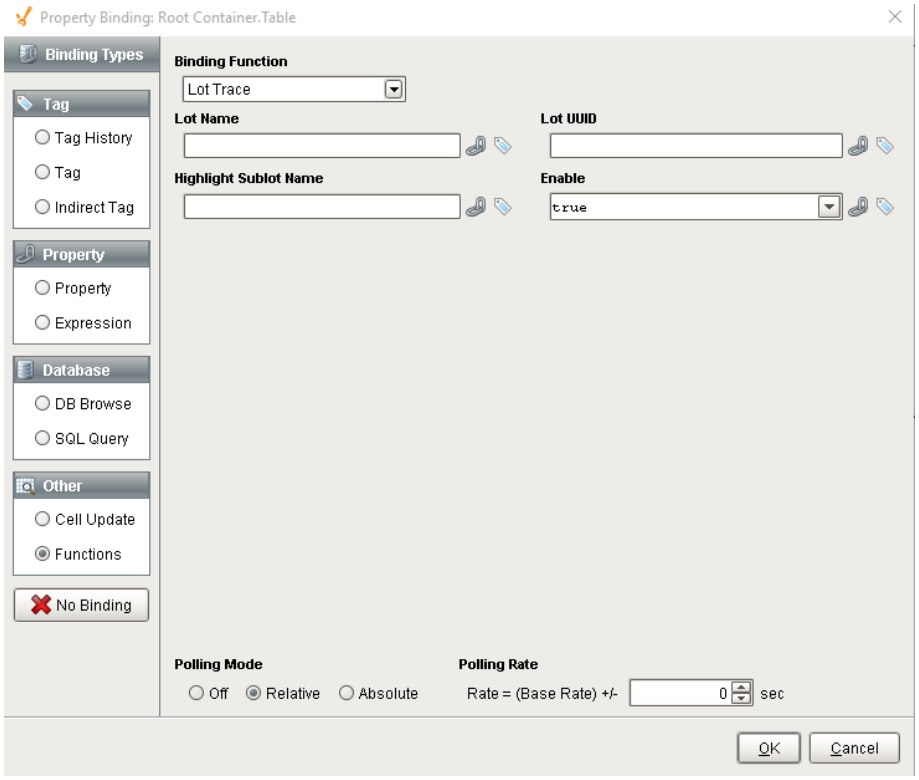
Parameters

Name	Description	Property Type
Lot UUID	The lot UUID to return the lot information results. Optionally Lot Name can be used but not both. See UUIDs for more information.	String
Lot Number	The lot number to return the material lot object for.	String
Lot Sequence	The lot sequence number to return the material lot object for. If it is less than zero, then the lot holding the maximum value is returned and if the same lot number is used for multiple segments, each lot with the same lot number will be assigned a different lot sequence number.	String

Lot Trace

Description

The Lot Trace binding function is used to retrieve trace information for a lot based on the filter parameters listed in the image below. It returns the same data that the Trace Graph component uses to display it's information.



Function Name

Lot Trace

Parameters

Name	Description	Property Type
Enable	If true, results will be retrieved. This provides a method to disable the retrieval of lot information results and the associated overhead.	Boolean
Lot Name	The lot name to return the trace details results. Optionally Lot UUID can be used but not both.	String
Lot UUID	The lot UUID to return the trace details results. Optionally Lot Name can be used but not both. See UUIDs for more information.	String
Highlight Sublot Name	If this property is not blank and a sublot of a lot name matches this property, then the LotContainsSublot column in the results will be set to 1. This indicates that the lot contains the specified sublot.	String

Lot Trace Results

The lot trace results are returned as a Dataset with the following columns.

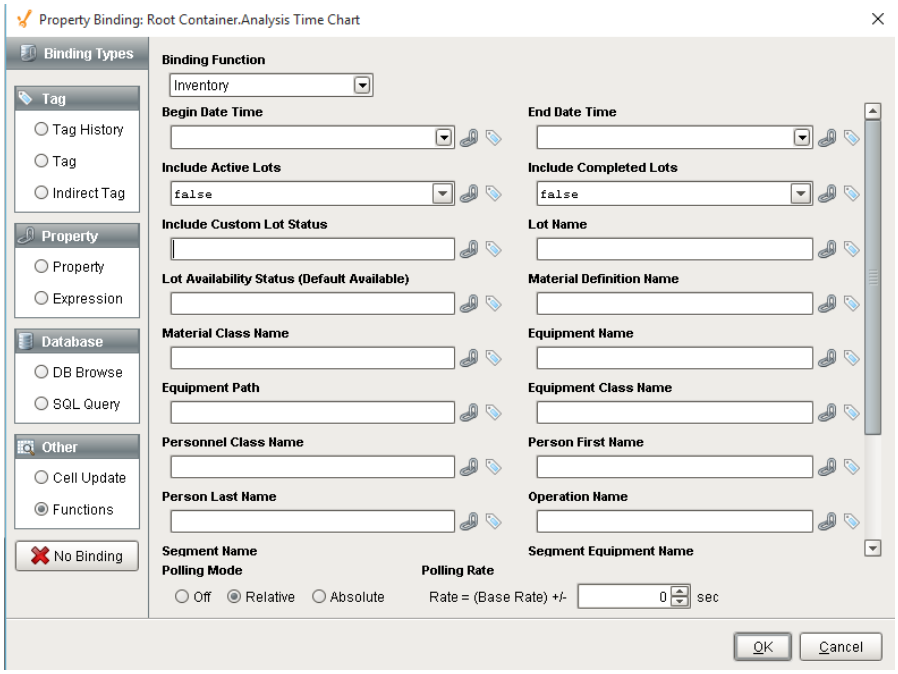
Column Name	Data Type
LotUUID	String
LotName	String
LotSequence	Integer
LotStatus	String
LotUse	String
LotBeginDateTime	Date
LotEndDateTime	Date
LotQuantity	Double
MaterialUUID	String
MaterialName	String
LotLocationUUID	String
LotLocationName	String
SegmentUUID	String
SegmentName	String
SegmentBeginDateTime	Date
SegmentEndDateTime	Date
SegmentLocationUUID	String

Column Name	Data Type
SegmentLocationName	String
PrevSegmentUUID	String
NextSegmentUUID	String
SegInCount	Integer
SegOutCount	Integer
LotInCount	Integer
LotOutCount	Integer
LotContainsSublot	Integer

Material Inventory

Description

The Inventory binding function is used to retrieve inventory information based on the filter parameters listed in the image below. It is an excellent method to get inventory of a particular material or class of material. But much more can be done by combining filters specified by the parameters below to zero in on the inventory information of interest.



Function Name

Inventory

Parameters

Name	Description	Property Type
Begin Date Time	When the Include Inactive Lots parameter is set to True, the results are limited to only include results that were processed at or after this property.	Calendar
End Date Time	When the Include Inactive Lots parameter is set to True, the results are limited to only include results that were processed at or before this property.	Calendar
Include Active Lots	If true, include active MES Material Lots or MES Material Sublots. Active items are those currently being processed.	Boolean
Include Completed Lots	If true, include completed MES Material Lots or MES Material Sublots.	Boolean

Name	Description	Property Type
Include Custom Lot Status	If the Final Lot Status setting of a material resource is set, then it can be filtered using this property. For example, MES Material Lot or MES Material Sublot objects can be set to Hold or any other value and then can be filtered here. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.	String
Lot Name	The lot name to return the trace details. Optionally Lot UUID can be used but not both.	String
Lot Availability Status (Default Available)	The availability of the lots can be filtered using this property. ***Options*** **Available** - The material lots that are currently available. **Used** - The material lots that are being used. **Both** - Results will contain both available and the used lots.	String
Material Definition Name	The results can be limited to only include lots or sublots that the associated material matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character. Only one of the Material Class Name, Material Class UUID, Material Definition Name, Material Definition UUID properties can be specified at a time. **Example:** *Balsamic*	String
Material Class Name	The results can be limited to only include lots or sublots that the associated material is included in a material class that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character. Only one of the Material Class Name, Material Class UUID, Material Definition Name, Material Definition UUID properties can be specified at a time. **Example:** Vinegar	String

Name	Description	Property Type
Equipment Name	The results can be limited to only include lots or sublots that are or were stored at the equipment that are included in a equipment class that match this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character. Only one of the EquipmentName, EquipmentUUID, or Equipment Path properties can be specified at a time. **Example:** Vinegar Tank 1	String
Equipment Path	The results can be limited to only include lots or sublots that are or were stored at the equipment that match this property. See Equipment for more information on equipment paths Only one of the Equipment Class Name, Equipment Class UUID, Equipment Path or Equipment UUID properties can be specified at a time. **Example:** My Enterprise\California\Storage\Vinegar Tanks\Vinegar Tank 1	String
Equipment Class Name	The results can be limited to only include lots or sublots that are or were stored at the equipment that are included in an equipment class that match this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character. Only one of the Equipment Class Name, Equipment Class UUID, Equipment Path or Equipment UUID properties can be specified at a time. **Example:** Vinegar Storage Tanks	String
Personnel Class Name	The results can be limited to only include lots or sublots that are or were handled by personnel that are included in a personnel class that match this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.	String

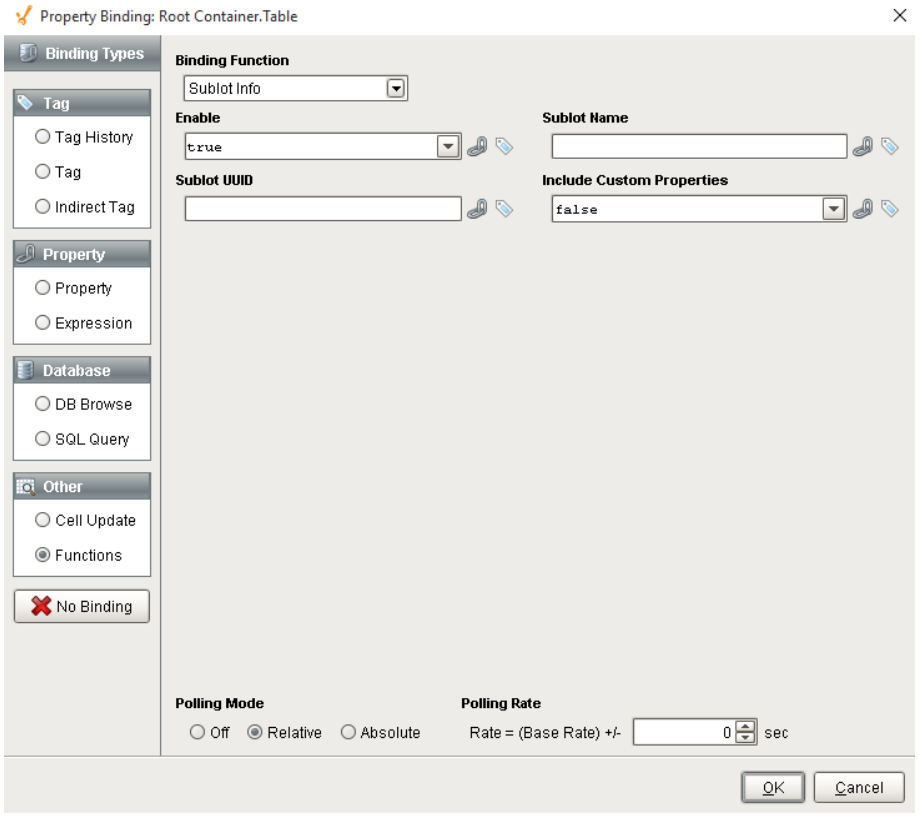
Name	Description	Property Type
	Only one of the Personnel Class Name, Personnel Class UUID or Person First Name and Person Last Name combination properties can be specified at a time. **Example:** Unload Operator	
Person First Name	The results can be limited to only include lots or sublots that are or were handled by personnel that match this property . It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.	String
Person Last Name	The results can be limited to only include lots or sublots that are or were handled by personnel that match this property . It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character.	String
Operation Name	The results can be limited to only include lots or sublots that performs the operation that match this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character. **Example:** Unload Vinegar	String
Segment Name	The results can be limited to only include lots or sublots that are handled by segment that match this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character. **Example:** Unload Balsamic Vinegar	String
Segment Equipment Name	The results can be limited to only include lots or sublots that are handled by equipment that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character. Only one of the Segment Equipment Class Name properties can be specified at a time.	String

Name	Description	Property Type
	Example: Bottling Line 1	
Segment Equipment Path	The results can be limited to only include lots or sublots that are or were stored at the segment equipment that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character. Only one of the Segment Equipment Class Name properties can be specified at a time. **Example:** Dressings Inc\California\Bottling\Bottling Line 1	String
Segment Equipment Class	The results can be limited to only include lots that were processed at the equipment that belong to the segment equipment class that matches this property. It can contain wildcard characters including * or ?. The * character can be any characters and the ? character represents any single character. Only one of the Segment Equipment Class Name properties can be specified at a time. **Example:** Bottling	String
Custom Property Values	The results can be limited to only include items that have a custom property expressions defined by this property that evaluates to true. **Example:** Kind > 3	PyDictionary
Limit Rows To (Default 1000)	The maximum number of samples to return in the results.	Integer

Sublot Info

Description

The Sublot Info binding function is used to retrieve information for a sublot based on the filter parameters listed in the image below. It can be used to display and report sublot details including custom property values.



Function Name

Sublot Info

Parameters

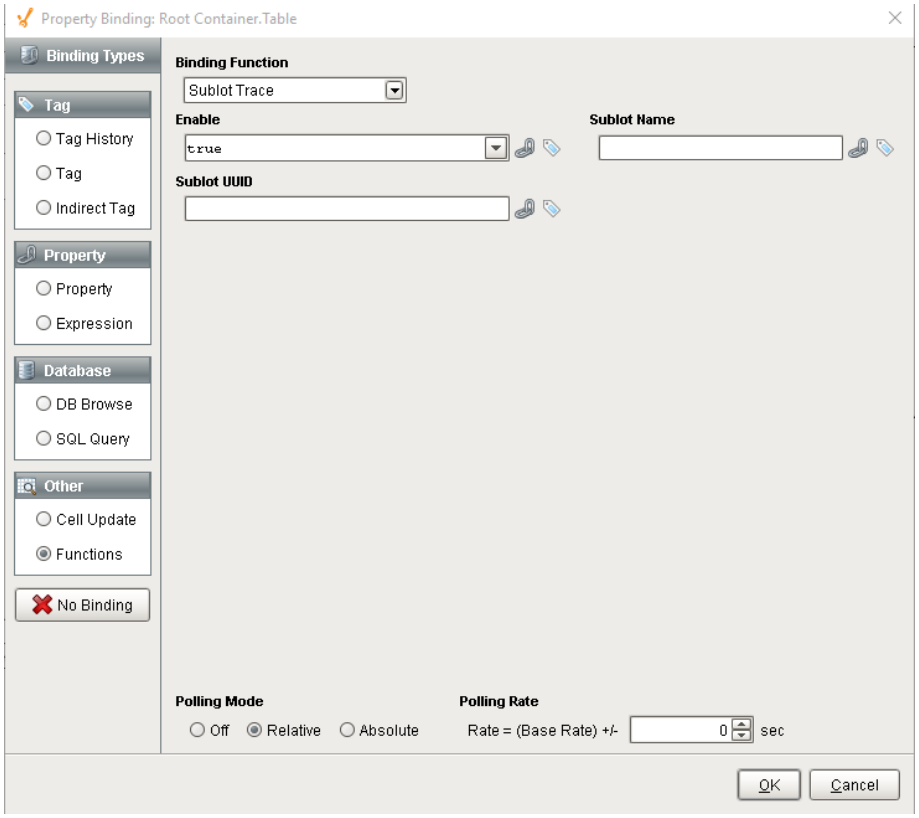
Name	Description	Property Type
Enable	If true, results will be retrieved. This provides a method to disable the retrieval of lot information results and the associated overhead.	Boolean
		String

Name	Description	Property Type
Sublot Name	The sublot name to return the sublot information results. Optionally Sublot UUID can be used but not both.	
Sublot UUID	The sublot UUID to return the sublot information results. Optionally sublot Name can be used but not both. See UUIDs for more information.	String
Include Custom Properties	If true, the results will include a column named CustomProperties that contains a Dataset to hold custom property information .	Boolean

Sublot Trace

Description

The Sublot Trace binding function is used to retrieve trace information for a sublot based on the filter parameters listed in the image below. It returns the same data that the Trace Graph component used to display it's information.



Function Name

Sublot Trace

Parameters

Name	Description	Property Type
Enable	If true, results will be retrieved. This provides a method to disable the retrieval of lot information results and the associated overhead.	Boolean
Sublot Name	The sublot name to return the trace details results. Optionally Sublot UUID can be used but not both.	String
Sublot UUID	The sublot UUID to return the trace details results. Optionally sublot Name can be used but not both. See UUIDs for more information.	String

Lot Trace Results

The sublot trace results are returned as a Dataset with the following columns.

Column Name	Data Type
LotUUID	String
LotName	String
LotSequence	Integer
LotStatus	String
LotUse	String
LotBeginDateTime	Date
LotEndDateTime	Date
LotQuantity	Double
MaterialUUID	String
MaterialName	String
LotLocationUUID	String
LotLocationName	String
SegmentUUID	String
SegmentName	String
SegmentBeginDateTime	Date
SegmentEndDateTime	Date
SegmentLocationUUID	String

Column Name	Data Type
SegmentLocationName	String
PrevSegmentUUID	String
NextSegmentUUID	String
SegInCount	Integer
SegOutCount	Integer
LotInCount	Integer
LotOutCount	Integer
LotContainsSublot	Integer

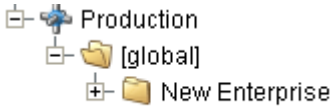
9.9 OPC Production Server Tag Reference

This references details the OPC values and child folders for node types that appear when browsing the Production OPC Server. For each property, the Ignition data type is listed and if it is read only. The Ignition data types correspond to the data types that are available for SQLTags.

Within this reference, **Read Only** means that the OPC value cannot be written to through the OPC Production Server. It can only be set in the designer or it is a calculated value. Trying to write to a read only property will result in an error message.

9.9.1 Project Tags

Each project within Ignition has its own production model. The first node(s) under the main Production node represent the Ignition project(s). Their names are the same as the project name. The image below represents the global project.



Project

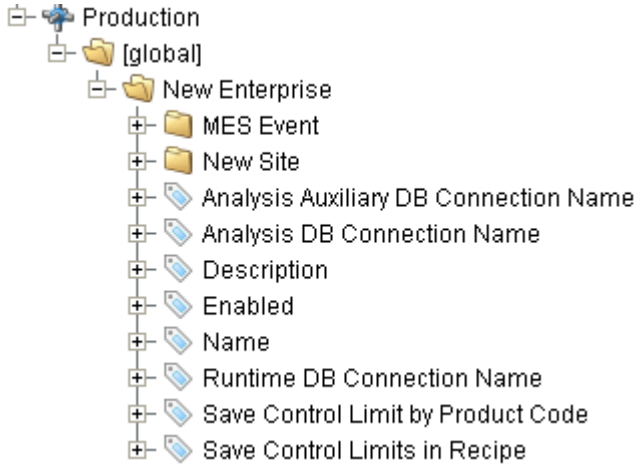
Child Folders

Enterprise	One folder will exist for each Enterprise that has been configured in the Ignition Designer. The folder can be opened to view all values within the enterprise.

9.9.2 Enterprise Tags

Description

The enterprise folder contains some properties associated with the enterprise, the folder for MES events and a folder for each production Site within it. The name is the same as the enterprise name that is configured in the designer. The image below represents the "New Enterprise" of the global project.



Enterprise

Child Folders

MES Event	MES Event folder has one folder for each MES Event (events can be added or edited in the general tab at the Enterprise level in the production model).
Site	One folder will exist for each Site that has been configured in the Ignition Designer. The folder can be opened to view all values within the site .

Properties

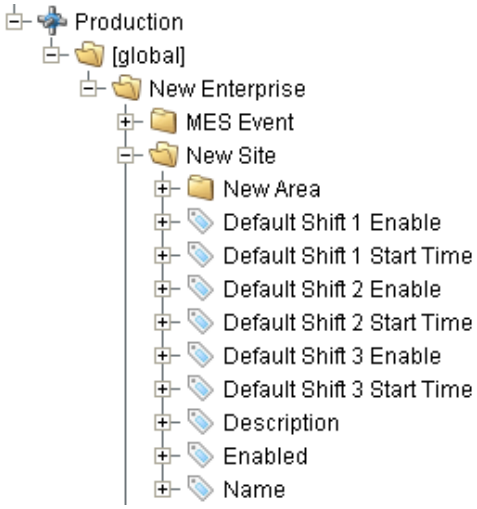
Analysis Auxiliary DB Connection Name	The name of the auxiliary (mirror) analysis database connection. Can be blank if no auxiliary DB connection is configured.	String Read Only
Analysis DB Connection Name	The name of the analysis database connection.	String Read Only
Description	Optionally, this property can be set to a description for the enterprise . It is not used by the OEE Downtime and Scheduling Module other than for reference.	String
Enabled	This reflects the enterprise Enabled property in the Designer. If the enterprise Enabled is set to true, then the OEE Downtime and Scheduling module will perform calculations for the enterprise and all sites , areas , lines and cells within it. If this property is set to false, then none of the sites , areas , lines or cells will have calculations performed.	Boolean
Name	This reflects the name of the enterprise that is set in the designer.	String Read Only
Runtime DB Connection Name	The name of the runtime database connection.	String Read Only
Save Control Limit by Product Code	Indicates if control limits are saved by product code. Exists only if the SPC module is also installed.	Boolean Read Only

Save Control Limits in Recipe	Indicates if control limits are saved by recipe. Exists only if the Recipe module is also installed.	Boolean Read Only

9.9.3 Site Tags

Description

The site folder contains some properties associated with the production site and a folder for each production area within it. The name is the same as the site name that is configured in the designer. The image below represents the "New Site" of the global project.



Site

Child Folders

Area	One folder will exist for each area that has been configured in the Ignition Designer. The folder can be opened to view all values within the area.

Properties

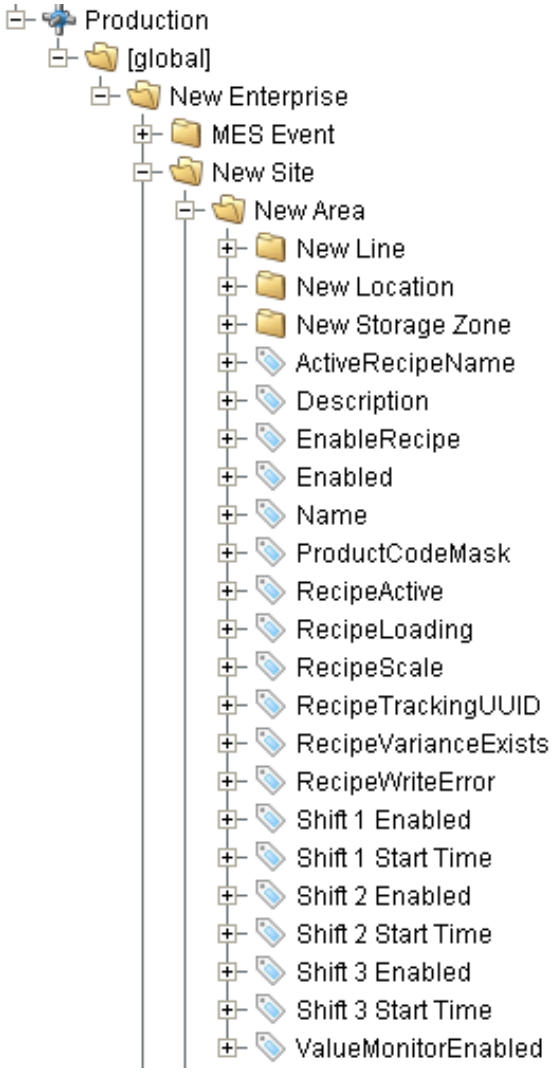
Description	Optionally, this property can be set to a description for the site. It is not used by the OEE Downtime and Scheduling Module other than for reference.	String

Enabled	This reflects the site Enabled property in the Designer. If the site Enabled is set to true, then the OEE Downtime and Scheduling module will perform calculations for the site and all areas, lines and cells within it. If this property is set to false, then none of the areas, lines or cells will have calculations performed.	Boolean
Name	This reflects the name of the site that is set in the designer.	String Read Only
Default Shift 1 Enable	This property will enable the shift 1 by default.	Boolean
Default Shift 1 Start Time	This reflects the site Default Shift 1 Start Time property in the Designer. See Site Configuration for more details.	DateTime Read Only
Default Shift 2 Enable	This property will enable the shift 2 by default.	Boolean
Default Shift 2 Start Time	This reflects the site Default Shift 2 Start Time property in the Designer. See Site Configuration for more details.	DateTime Read Only
Default Shift 3 Enable	This property will enable the shift 3 by default.	Boolean
Default Shift 3 Start Time	This reflects the site Default Shift 3 Start Time property in the Designer. See Site Configuration for more details.	DateTime Read Only

9.9.4 Area Tags

Description

The area folder contains some properties associated with the production area and folders for each production line, location and storage zone within it. The name is the same as the area name that is configured in the designer. The image below represents the "New Area" of the global project.



Area

Child Folders

Line	One folder will exist for each line that has been configured in the Ignition Designer. The folder can be opened to view all values within the line.

Location	One folder will exist for each location that has been configured in the Ignition Designer. The folder can be opened to view all values within the location.
Storage Zone	One folder will exist for each storage zone that has been configured in the Ignition Designer. The folder can be opened to view all values within the storage zone.

Properties

ActiveRecipeName	If a recipe is active for this production area, then this is the name of the recipe. If a recipe is not active, then this is blank.	String Read Only
Description	Optionally, this property can be set to a description for the area . It is not used by the OEE Downtime and Scheduling Module other than for reference.	String
EnableRecipe	Set to true to allow recipes to be selected for this production item. This is useful when selecting a recipe for a production area and preventing selecting the same recipe for selected child production items.	Boolean
Enabled	This reflects the site Enabled property in the Designer. If the area Enabled is set to true, then the OEE Downtime and Scheduling module will perform calculations for the area and all lines and cell within it. If this property is set to false, then none of the lines or cells will have calculations performed.	Boolean
Name	This reflects the name of the area that is set in the designer.	String Read Only
ProductCodeMask		
RecipeActive	Indicates if a recipe is currently active.	Boolean Read Only

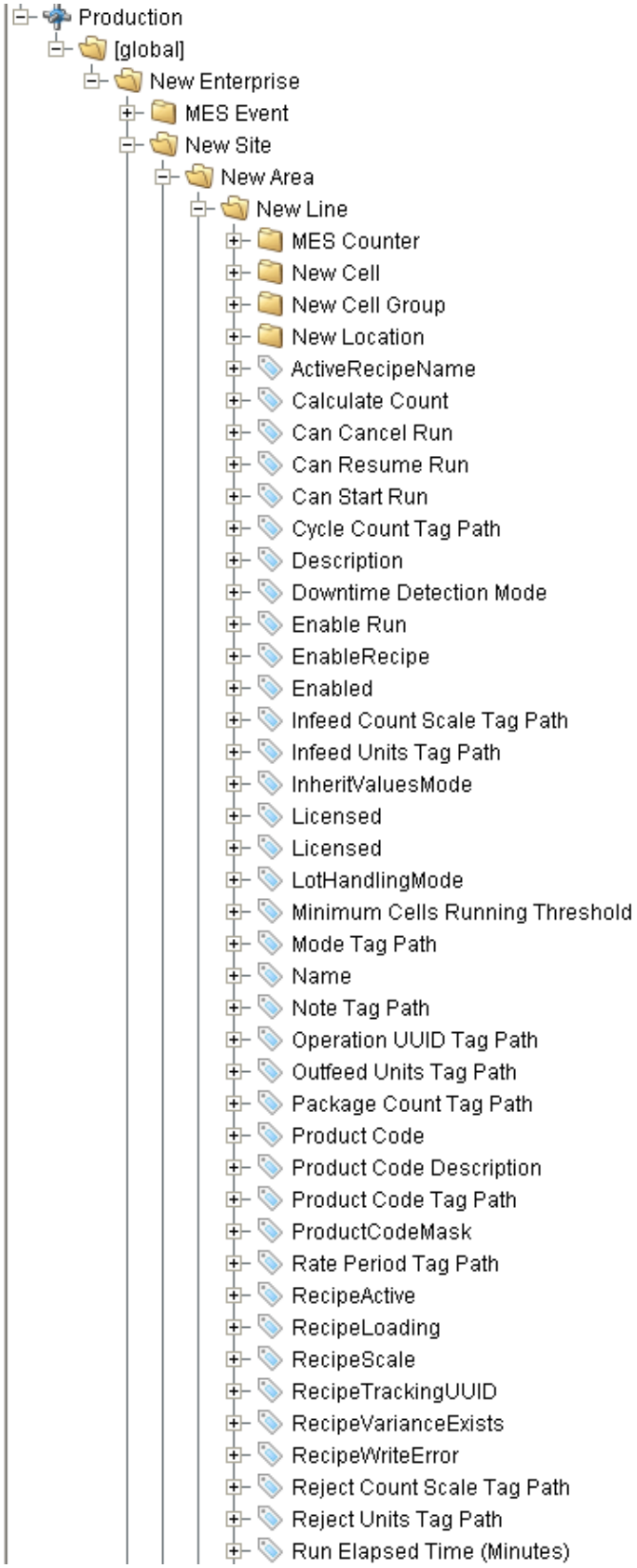
RecipeLoading	True if a recipe is currently being loaded for the production area.	Boolean Read Only
RecipeScale	Set this to the amount to scale a recipe prior to selecting a recipe for the production area.	Double
RecipeTrackingUUID	This is a unique value used for tracking initial recipe values and variances while a recipe is selected. It can be used when looking up data directly from the database.	String Read Only
RecipeVarianceExists	If true, then Ignition tags associated with at least one recipe value for this production item have changed.	Boolean Read Only
RecipeWriteError	If true, then at least one recipe value did not write to the associated Ignition tags when the recipe was first selected.	Boolean Read Only
Shift 1 Enabled	This property will enable shift 1.	Boolean
Shift 1 Start Time	The current Shift 1 Start Time time for the production area . If the associated Shift 1 Start Time property for the area in the designer is set to Inherit From Parent , this will be the time defined for the parent production site . See Area Configuration for more details.	DateTime Read Only
Shift 2 Enabled	This property will enable shift 2.	Boolean
Shift 2 Start Time	The current Shift 2 Start Time time for the production area . If the associated Shift 2 Start Time property for the area in the designer is set to Inherit From Parent , this will be the time defined for the parent production site . See Area Configuration for more details.	DateTime Read Only
Shift 3 Enabled	This property will enable shift 3.	Boolean

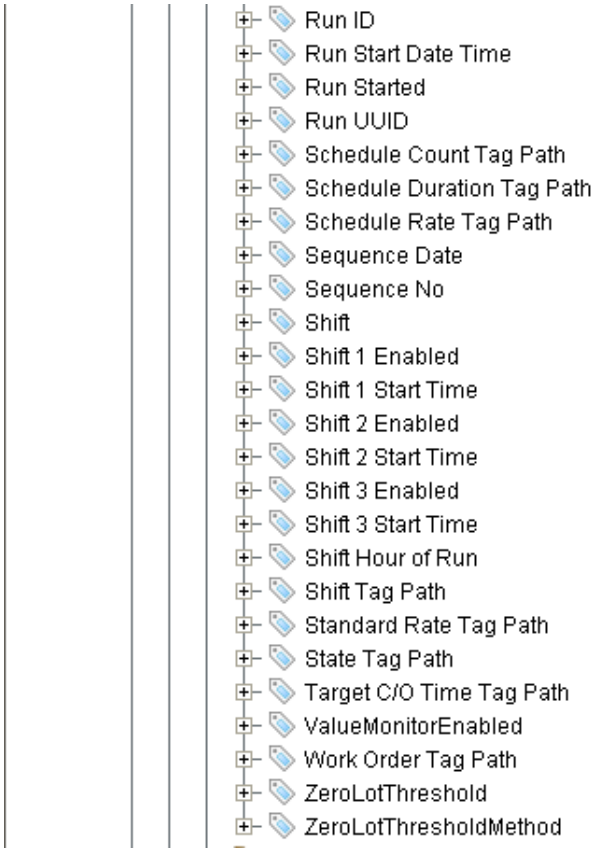
Shift 3 Start Time	The current Shift 3 Start Time time for the production area . If the associated Shift 3 Start Time property for the area in the designer is set to Inherit From Parent , this will be the time defined for the parent production site . See Area Configuration for more details.	DateTime Read Only
ValueMonitorEnabled	If true, recipe values are being monitored and recipe value variances will be logged.	Boolean Read Only

9.9.5 Line Tags

Description

The line folder contains some properties associated with the production line, the folder for MES counter and folders for each production cell, cell group, location within it. The name is the same as the line name that is configured in the designer. The image below represents the "New Line" of the global project.


Production
[global]
New Enterprise
MES Event
New Site
New Area
New Line
MES Counter
New Cell
New Cell Group
New Location
ActiveRecipeName
Calculate Count
Can Cancel Run
Can Resume Run
Can Start Run
Cycle Count Tag Path
Description
Downtime Detection Mode
Enable Run
EnableRecipe
Enabled
Infeed Count Scale Tag Path
Infeed Units Tag Path
InheritValuesMode
Licensed
Licensed
LotHandlingMode
Minimum Cells Running Threshold
Mode Tag Path
Name
Note Tag Path
Operation UUID Tag Path
Outfeed Units Tag Path
Package Count Tag Path
Product Code
Product Code Description
Product Code Tag Path
ProductCodeMask
Rate Period Tag Path
RecipeActive
RecipeLoading
RecipeScale
RecipeTrackingUUID
RecipeVarianceExists
RecipeWriteError
Reject Count Scale Tag Path
Reject Units Tag Path
Run Elapsed Time (Minutes)



Line

Child Folders

MES Counter	One folder will exist for each MES Counter that has been configured in the Ignition Designer. The folder can be opened to view all values within the counter.
Cell	One folder will exist for each cell that has been configured in the Ignition Designer. The folder can be opened to view all values within the cell.
Cell Group	One folder will exist for each cell group that has been configured in the Ignition Designer. The folder can be opened to view all values within the cell group.
Location	One folder will exist for each location that has been configured in the Ignition Designer. The folder can be opened to view all values within the location.

Properties

ActiveRecipeName	If a recipe is active for this production line, then this is the name of the recipe. If a recipe is not active, then this is blank.	String Read Only
Calculate Count	This value will increment every time OEE, downtime and scheduling values are calculated for the project production model.	Int4 Read Only
Can Cancel Run	Indicates if this run can be cancelled. Runs can only be cancelled while in changeover	Boolean Read Only
Can Resume Run	If true, all conditions are good to resume a production run.	Boolean Read Only
Can Start Run	If true, all conditions are good to start a production run.	Boolean Read Only
Cycle Count Tag Path	Tag path to read the current equipment cycle count from.	String
Description	Optionally, this property can be set to a description for the line. It is not used by the OEE Downtime and Scheduling Module other than for reference.	String
Downtime Detection Mode	This reflects the current value of the "Downtime Detection Method" setting in the designer.	String Read Only
Enable Run		Boolean

	Setting Enable Run to true will enable the production run for the line. Setting it to false will end the production run. Typically, this is controlled by the functionality of the operator screen, but it can also be handled programmatically.	
EnableRecipe	Set to true to allow recipes to be selected for this production item. This is useful when selecting a recipe for a production line and preventing selecting the same recipe for selected child production items.	Boolean
Enabled	This reflects the line Enabled property in the Designer. If the line Enabled is set to true, then the OEE Downtime and Scheduling module will perform calculations for the all cells within it. If this property is set to false, then none of the cells will have calculations performed.	Boolean
Infeed Count Scale Tag Path	Tag path to read the current equipment infeed count scale from.	String
Infeed Units Tag Path	Tag path to read the current infeed units.	String
InheritValuesMode		
Licensed	This reflects whether the modules are licensed or not.	Boolean
Lot Handling Mode	The lot handling mode type.	String
Minimum Cells Running Threshold	For Parallel downtime detection method, the minimum number of cells that must be running before the group is considered down.	Integer
Mode Tag Path	Tag path to read the current equipment mode from.	String
Name	This reflects the name of the line that is set in the designer.	String Read Only

Note Tag Path	Tag path to read the current downtime note from.	String
Operation UUID Tag Path	Tag path to read the current MES Operations Response UUID from.	String
Outfeed Units Tag Path	Tag path to read the current outfeed units.	String
Package Count Tag Path	Tag path to read the current equipment package count from.	String
Product Code	The current product code being run on the line. Typically, this is controlled by the functionality of the operator screen, but it can also be handled programmatically. It should only be changed when Enable Run is false.	String
Product Code Description	The description for the current Product Code.	String Read Only
Product Code Tag Path	Tag path to read the current equipment product code from.	String
ProductCodeMask		
Rate Period Tag Path	Tag path to read the rate period code from.	String
RecipeActive	Indicates if a recipe is currently active.	Boolean Read Only
RecipeLoading	True if a recipe is currently being loaded for the production line.	Boolean Read Only
RecipeScale	Set this to the amount to scale a recipe prior to selecting a recipe for the production line.	Double

RecipeTrackingUUID	This is a unique value used for tracking initial recipe values and variances while a recipe is selected. It can be used when looking up data directly from the database.	String Read Only
RecipeVarianceExists	If true, then Ignition tags associated with at least one recipe value for this production item have changed.	Boolean Read Only
RecipeWriteError	If true, then at least one recipe value did not write to the associated Ignition tags when the recipe was first selected.	Boolean Read Only
Reject Count Scale Tag Path		
Reject Units Tag Path	Tag path to read the current reject units.	String
Run Elapsed Time (Minutes)	The total minutes that have elapsed from the start of the production run.	Float8 Read Only
Run ID	This is the unique identification number that was generated by the database when a row is inserted into the Run table. It can be used to associate external data to a production run.	Int4 Read Only
Run Start Date Time	This will equal the time that the production run started or the beginning of the current shift, whichever occurred last.	DateTime Read Only
Run Started	The value will be true if a production run has started. Even if the production run has been ended but a new production run has not been selected, this value will be true.	Boolean Read Only
Run UUID		

Schedule Count Tag Path	Tag path to read the schedule count from.	String
Schedule Duration Tag Path	Tag path to read the schedule duration from.	String
Schedule Rate Tag Path	Tag path to read the target schedule rate from.	String
Sequence Date	The date and time that the current shift started. This is used for retrieving results based on a production day and not days that are split at midnight.	Date Read Only
Sequence No	A number that is 0 at the beginning of a production run and increments at the beginning of every shift.	Int4 Read Only
Shift	The current shift based on the shift start times configured for the production line.	Int4 Read Only
Shift 1 Enabled	The current Shift 1 enabled state for the production line. It reflects the Shift 1 Enabled property for the line in the designer. The initial value of this property is determined by the Shift 1 Initial Enabled State property for the production line in the designer. It can be changed from the initial value.	Boolean
Shift 1 Start Time	The current Shift 1 Start Time time for the production line. If the associated Shift 1 Start Time property for the line in the designer is set to Inherit From Parent, this can be the time defined for the parent production area or line.	DateTime
Shift 2 Enabled	The current Shift 2 enabled state for the production line. It reflects the Shift 2 Enabled property for the line in the designer. The initial value of this property is determined by the Shift 2 Initial Enabled State property for the production line in the designer. It can be changed from the initial value	Boolean

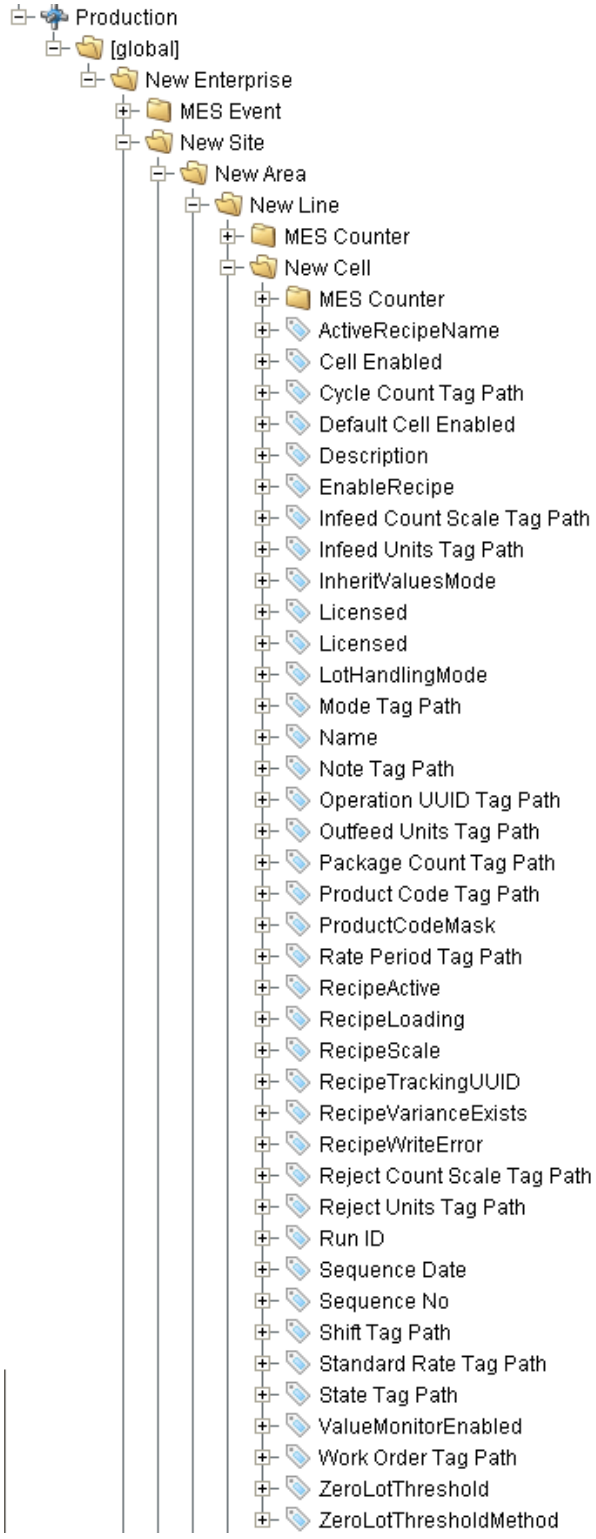
Shift 2 Start Time	The current Shift 2 Start Time time for the production line. If the associated Shift 2 Start Time property for the line in the designer is set to Inherit From Parent, this can be the time defined for the parent production area or line.	DateTime
Shift 3 Enabled	The current Shift 3 enabled state for the production line. It reflects the Shift 3 Enabled property for the line in the designer. The initial value of this property is determined by the Shift 3 Initial Enabled State property for the production line in the designer. It can be changed from the initial value.	Boolean
Shift 3 Start Time	The current Shift 3 Start Time time for the production line. If the associated Shift 3 Start Time property for the line in the designer is set to Inherit From Parent, this can be the time defined for the parent production area or line.	DateTime
Shift Hour Of Run		
Shift Tag Path	Tag path to read the current equipment production shift from.	String
Standard Rate Tag Path	Tag path to read the current equipment standard rate from.	String
State Tag Path	Tag path to read the current equipment state from.	String
Target C/O Time Tag Path	Tag path to read the target changeover time from.	String
ValueMonitorEnabled	If true, recipe values are being monitored and recipe value variances will be logged.	Boolean Read Only
Work Order Tag Path	Tag path to read the current work order from.	String
ZeroLotThreshold		

ZeroLotThresholdMethod		

9.9.6 Cell Tags

Description

The cell folder contains some properties associated with the production cell and the MES Counter folder. The name is the same as the cell name that is configured in the designer. The image below represents the 'New Cell' of the global project.


Production
[global]
New Enterprise
MES Event
New Site
New Area
New Line
MES Counter
New Cell
MES Counter
ActiveRecipeName
Cell Enabled
Cycle Count Tag Path
Default Cell Enabled
Description
EnableRecipe
Infeed Count Scale Tag Path
Infeed Units Tag Path
InheritValuesMode
Licensed
Licensed
LotHandlingMode
Mode Tag Path
Name
Note Tag Path
Operation UUID Tag Path
Outfeed Units Tag Path
Package Count Tag Path
Product Code Tag Path
ProductCodeMask
Rate Period Tag Path
RecipeActive
RecipeLoading
RecipeScale
RecipeTrackingUUID
RecipeVarianceExists
RecipeWriteError
Reject Count Scale Tag Path
Reject Units Tag Path
Run ID
Sequence Date
Sequence No
Shift Tag Path
Standard Rate Tag Path
State Tag Path
ValueMonitorEnabled
Work Order Tag Path
ZeroLotThreshold
ZeroLotThresholdMethod

Cell

Child Folders

MES Counter	One folder will exist for each MES Counter that has been configured in the Ignition Designer. The folder can be opened to view all values within the counter.

Properties

ActiveRecipeName	If a recipe is active for this production cell, then this is the name of the recipe. If a recipe is not active, then this is blank.	String Read Only
Cell Enabled	If Cell Enabled is set to true, then the OEE Downtime and Scheduling module will perform calculations for the cell. This value is determined by the product code and production line. It can also be programmatically changed.	Boolean Read Only
Cycle Count Tag Path	Tag path to read the current equipment cycle count from.	String
Default Cell Enabled	This reflects the site Default Cell Enabled property in the Designer.	Boolean Read Only
Description	Optionally, this property can be set to a description for the cell. It is not used by the OEE Downtime and Scheduling Module other than for reference.	String
EnableRecipe	Set to true to allow recipes to be selected for this production item. This is useful when selecting a recipe for a production cell and preventing selecting the same recipe for selected child production items.	Boolean
Infeed Count Scale Tag Path		
Infeed Units Tag Path	Tag path to read the current infeed units.	String

InheritValuesMode		
Licensed	This reflects whether the modules are licensed or not.	Boolean
LotHandlingMode	The lot handling mode type.	String
Mode Tag Path	Tag path to read the current equipment mode from.	String
Name	This reflects the name of the cell that is set in the designer.	String Read Only
Note Tag Path	Tag path to read the current downtime note from.	String
Operation UUID Tag Path	Tag path to read the current MES Operations Response UUID from.	String
Outfeed Units Tag Path	Tag path to read the current outfeed units.	String
Package Count Tag Path	Tag path to read the current equipment package count from.	String
Product Code Tag Path	Tag path to read the current equipment product code from.	String
ProductCodeMask		String
Rate Period Tag Path	Tag path to read the rate period code from.	String
RecipeActive	Indicates if a recipe is currently active.	Boolean Read Only
RecipeLoading	True if a recipe is currently being loaded for the production cell.	Boolean Read Only
RecipeScale		Double

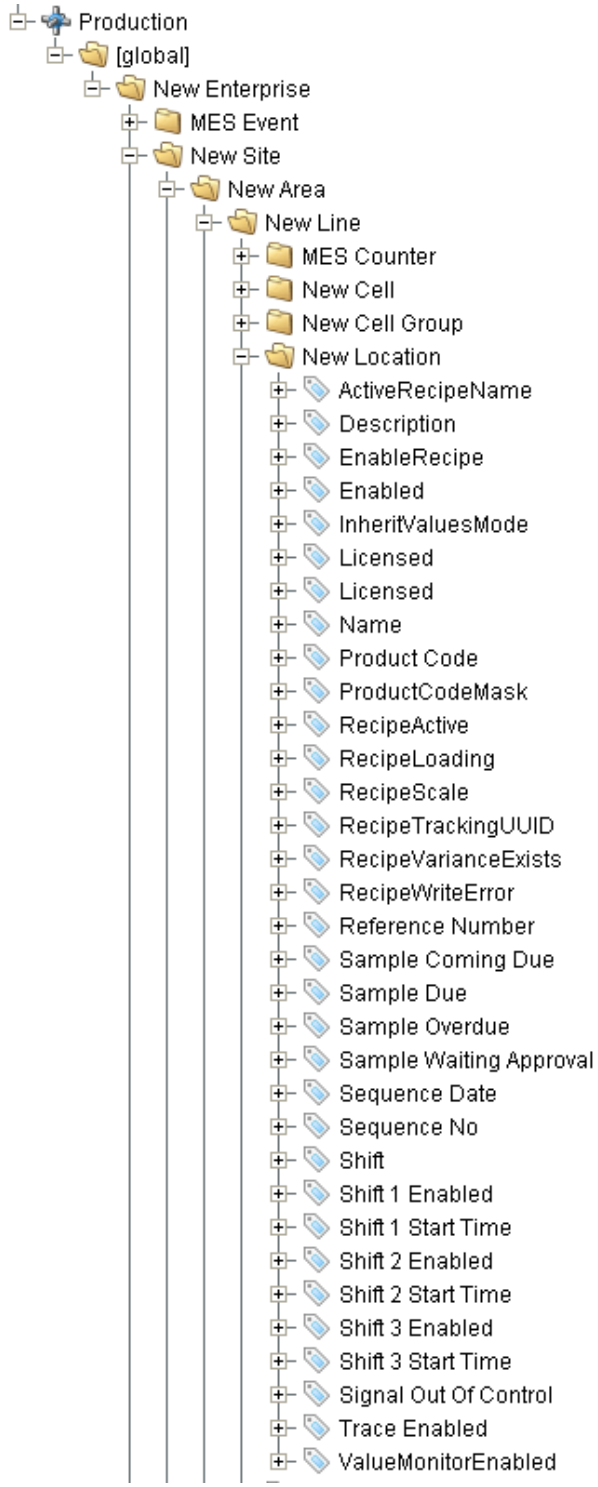
	Set this to the amount to scale a recipe prior to selecting a recipe for the production cell.	
RecipeTrackingUUID	This is a unique value used for tracking initial recipe values and variances while a recipe is selected. It can be used when looking up data directly from the database.	String Read Only
RecipeVarianceExists	If true, then Ignition tags associated with at least one recipe value for this production item have changed.	Boolean Read Only
RecipeWriteError	If true, then at least one recipe value did not write to the associated Ignition tags when the recipe was first selected.	Boolean Read Only
Reject Count Scale Tag Path		
Reject Units Tag Path	Tag path to read the current reject units.	String
Run ID	This is the unique identification number that was generated by the database when a row is inserted into the Run table. It can be used to associate external data to a production run.	Int4 Read Only
Sequence Date	The date and time that the current shift started. This is used for retrieving results based on a production day and not days that are split at midnight.	Date Read Only
Sequence No	A number that is 0 at the beginning of a production run and increments at the beginning of every shift.	Int4 Read Only
Shift Tag Path	Tag path to read the current equipment production shift from.	String
Standard Rate Tag Path		String

	Tag path to read the current equipment standard rate from.	
State Tag Path	Tag path to read the current equipment state from.	String
ValueMonitorEnabled	If true, recipe values are being monitored and recipe value variances will be logged.	Boolean Read Only
Work Order Tag Path	Tag path to read the current work order from.	String
ZeroLotThreshold		
ZeroLotThresholdMethod		

9.9.7 Location Tags

Description

The location folder contains properties associated with the production location. The production location can reside under a production line or directly under a production area. The name is the same as the location name that is configured in the designer. The image below represents the 'New Location' of the global project.



Location

Child Folders

No child folders

Properties

ActiveRecipeName	If a recipe is active for this production location, then this is the name of the recipe. If a recipe is not active, then this is blank.	String Read Only
Description	Optionally, this property can be set to a description for the location. It is not used by the SPC Module other than for reference.	String
EnableRecipe	Set to true to allow recipes to be selected for this production item. This is useful when selecting a recipe for a production line and preventing selecting the same recipe for selected child production items.	Boolean
Enabled	If Enabled is set to true, then the SPC module will perform calculations and enable tag collectors for the location.	Boolean
InheritValuesMode		
Licensed	This reflects whether the modules are licensed or not.	Boolean
Name	This reflects the name of the location that is set in the designer.	String Read Only
Product Code	This reflects the product code currently assigned to this location.	String Read Only
ProductCodeMask		
RecipeActive	Indicates if a recipe is currently active.	Boolean Read Only
RecipeLoading	True if a recipe is currently being loaded for the production location.	Boolean

		Read Only
RecipeScale	Set this to the amount to scale a recipe prior to selecting a recipe for the production location.	Double
RecipeTrackingUUID	This is a unique value used for tracking initial recipe values and variances while a recipe is selected. It can be used when looking up data directly from the database.	String Read Only
RecipeVarianceExists	If true, then Ignition tags associated with at least one recipe value for this production item have changed.	Boolean Read Only
RecipeWriteError	If true, then at least one recipe value did not write to the associated Ignition tags when the recipe was first selected.	Boolean Read Only
Reference Number	This reflects the reference number currently assigned to this location. The reference number is optional and can represent anything you want to be tracked with samples at this location, except for the product code.	String Read Only
Sample Coming Due	If true, a sample is coming due for this location.	Boolean Read Only
Sample Due	If true, a sample is due for this location.	Boolean Read Only
Sample Overdue	If true, a sample is overdue for this location.	Boolean Read Only
Sample Waiting Approval	If true, an unapproved sample is waiting to be approved for this location.	Boolean

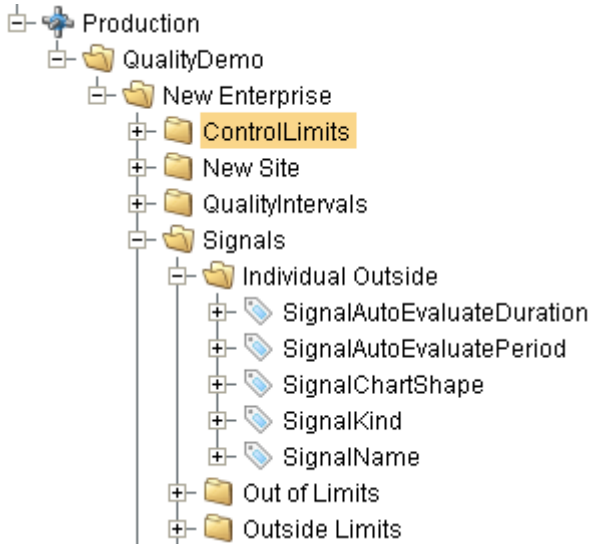
		Read Only
Sequence Date	The date and time that the current shift started. This is used for retrieving results based on a production day and not days that are split at midnight.	Date Read Only
Sequence No	A number that is 0 at the beginning of a production run and increments at the beginning of every shift.	Int4 Read Only
Shift	The current shift based on the shift start times configured for the production location.	Int4 Read Only
Shift 1 Enabled	The current Shift 1 enabled state for the production location. It reflects the Shift 1 Enabled property for the location in the designer. The initial value of this property is determined by the Shift 1 Initial Enabled State property for the production location in the designer. It can be changed from the initial value.	Boolean
Shift 1 Start Time	The current Shift 1 Start Time time for the production location. If the associated Shift 1 Start Time property for the location in the designer is set to Inherit From Parent, this can be the time defined for the parent production area or line.	DateTime
Shift 2 Enabled	The current Shift 2 enabled state for the production location. It reflects the Shift 2 Enabled property for the location in the designer. The initial value of this property is determined by the Shift 2 Initial Enabled State property for the production location in the designer. It can be changed from the initial value	Boolean
Shift 2 Start Time		DateTime

	The current Shift 2 Start Time time for the production location. If the associated Shift 2 Start Time property for the location in the designer is set to Inherit From Parent, this can be the time defined for the parent production area or line.	
Shift 3 Enabled	The current Shift 3 enabled state for the production location. It reflects the Shift 3 Enabled property for the location in the designer. The initial value of this property is determined by the Shift 3 Initial Enabled State property for the production location in the designer. It can be changed from the initial value.	Boolean
Shift 3 Start Time	The current Shift 3 Start Time time for the production location. If the associated Shift 3 Start Time property for the location in the designer is set to Inherit From Parent, this can be the time defined for the parent production area or line.	DateTime
Signal Out Of Control	If true, at least one signal associated with this location is out of control.	Boolean Read Only
Trace Enabled	If true, a product code has been assigned to this location and is considered as actively processing.	Boolean Read Only
ValueMonitorEnabled	If true, recipe values are being monitored and recipe value variances will be logged.	Boolean Read Only

9.9.8 Signals

Description

The signals folder contains a folder for each signal. The name of each folder is the same as the signal name that is configured in the designer. The image below represents the **Individual Outside** signal of the QualityDemo project.



SPCOPCSignals

Out of Control Signals

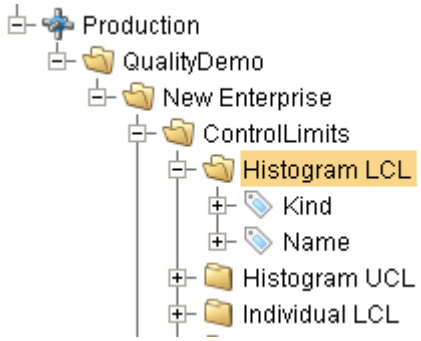
Properties

Kind	The ordinal value of the kind of control chart that the signal is associated with. See SignalKindTypes for more information.	int Read Only
SignalName	This reflects the name of the signal that is configured in the designer.	String Read Only
SignalAutoEvaluatePeriod	This reflects the ordinal value of the evaluation time period of the SignalAutoEvaluateDuration value. See Signal Auto Evaluate Period Types for more information.	int Read Only
SignalAutoEvaluateDuration	This reflects the duration to use when automatically evaluating sample data for a location for this signal.	int Read Only
SignalChartShape	This reflects the ordinal value of the shape to display in the control charts when a sample is out of control for this signal. See SPCChartShapeTypes for more information.	int Read Only

9.9.9 ControlLimits

Description

The control limits folder contains a folder for each control limit. The name of each folder is the same as the control limit name that is configured in the designer. The image below represents the **Histogram LCL** control limit of the QualityDemo project.



SPCOPCLimit

Control Limits

Properties

Kind	The ordinal value of the kind of control chart that the control limit is associated with. See ControlLimitKindTypes for more information.	int Read Only
Name	This reflects the name of the control limit that is configured in the designer.	String Read Only

10 Appendix B: Knowledge Base Articles

10.1 General

10.2 Track & Trace

10.3 OEE

10.4 SPC

10.5 Recipe

10.6 Web Services

11 Videos

These are the links to videos.

11.1 MES Basics

Installing Modules

What is MES

Module Installation

Configuring MES Databases

Installing Sample Project

Production Simulator

Understanding MES and Architectures

Standard Architecture

Enterprise Administration

Redundant Architecture

11.2 Recipe/Changeover

Understanding Recipe and Changeover

Recipe Management Overview

Configuring the Production Model

Configuring Tags

Configuring Production Model

Configuring Variance Logging

Production Tags

Scaling Recipe Setpoints

Configuring Recipes

Recipe Editor Overview

Recipe Editor – Default Values and Security

Recipe Editor – Adding Recipes

Recipe Editor – Master Recipes and Descendants

Recipe Editor – Sub Recipes

Recipe Editor – Notes

Recipe Editor – Filtering

Recipe Editor – Allowing Certain Functions

Recipe Editor Table

Loading Recipes

Manual – Component

Manual – Scripting

Automatic – Scripting

Analysis and Reports

Changelog

Variance Log

11.3 Instrument Interface

Instrument Interface Configurations

Instrument Interface Overview

Serial Settings

File Monitor Settings

11.4 OEE/Downtime

11.5 SPC

Understanding Quality and SPC

Quality Overview

Quality vs. SPC

SPC Variation

SPC Causes of Variation

SPC Samples

SPC Values and Attributes

SPC Standard Deviation

SPC Control Charts

Control Chart Introduction

Control Limit Introduction

SPC Rule Introduction

Individual Chart

Median Chart

XBarS Chart

Histogram Chart

Process Capability Charts

P and NP Charts

C and U Charts

Pareto Chart

Configuring the Production Model

Configuring Tags

Configuring Production Model

Control Limit, Rule, and Interval

Configuring Additional Factors

Production Tags

Operator Screens

Sample Definition Manager Overview

Manual Sample Entry List

Approving Samples

Sample Definition Location Ownership

Automatic Multipoint Sample Entry

Adding Sample in Scripting

Schedule Sample List

Schedule Monitoring

Building a Basic Control Chart

Rule Monitoring

SPC Analysis

Impromptu Control Chart Screen

Impromptu Analysis Screen

Using SPC Data in Reports

11.6 Web Services

Web Services Overview

Intro to Web Services

Module Overview

Create and Use a Web Service Configuration

12 Tips for Troubleshooting

13 Glossary

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Search

13.1 A

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

13.1.1 Anderson Darling Test

The **Anderson–Darling test** is a statistical test of whether a given sample of data is drawn from a given probability distribution . In its basic form, the test assumes that there are no parameters to be estimated in the distribution being tested, in which case the test and its set of critical values is distribution-free. However, the test is most often used in contexts where a family of distributions is being tested, in which case the parameters of that family need to be estimated and account must be taken of this in adjusting either the test-statistic or its critical values. When applied to testing whether a normal distribution adequately describes a set of data, it is one of the most powerful statistical tools for detecting most departures from normality. -wikipedia

13.1.2 API

Application **P**rogramming **I**nterface (**API**), in the context of Java, is a collection of prewritten packages, classes, and interfaces with their respective methods, fields, and constructors. Similar to a user interface, which facilitates interaction between humans and computers, an API serves as a software program interface facilitating interaction. -techopedia

13.2 B

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

13.3 C

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

13.3.1 CD-Key

A product key, also known as a software key, is a specific software-based key for a computer program. It certifies that the copy of the program is original.

When the software and the modules are purchased, you are provided with a six-digit CD-Key. If you add a module, your account is updated, and you can re-use your existing CD-Key to activate the new features. You can also unactivate your CD-Key, and reuse it to activate Ignition on a different machine.

13.3.2 CSV

A comma-separated values (**CSV**) file stores tabular data (numbers and text) in plain text. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format. -Wikipedia

13.4 D

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

13.4.1 Data Point

A **data point** is a discrete unit of information. In a general sense, any single fact is a **data point** . In a statistical or analytical context, a **data point** is usually derived from a measurement or research and can be represented numerically and/or graphically.

The measurements contained in a data point are formally typed, where here type is used in a way compatible with datatype in computing; so that the type of measurement can specify whether the measurement results in a Boolean value from {yes, no}, an integer or real number, the identity of some category, or some vector or array. The implication of point is often that the data may be plotted in a graphic display, but in many cases the data are processed numerically before that is done. In the context of statistical graphics, measured values for individuals or summary statistics for different subpopulations are displayed as separate symbols within a display; since such symbols can differ by shape, size and colour, a single data point within a display can convey multiple aspects of the set of measurements for an individual or subpopulation. -Wikipedia.

13.5 E

13.5.1 EAN

An EAN-13 barcode (originally European Article Number, but now renamed International Article Number even though the abbreviation EAN has been retained) is a 13 digit (12 data and 1 check) barcoding standard which is a superset of the original 12-digit Universal Product Code (UPC) system. The EAN-13 barcode is defined by the standards organization GS1.

The 13 digits in the EAN-13 barcode are grouped as follows:

- The left group: Digits 2-7. The left group also encodes digit 1, through a scheme of odd and even parity.
- The right group: Digits 8-13, digit 13 is the check digit. -Wikipedia.

13.5.2 ERP

Enterprise resource planning (ERP) is business management software—typically a suite of integrated applications—that a company can use to collect, store, manage and interpret data from many business activities.

13.6 F

13.6.1 EAN

An EAN-13 barcode (originally European Article Number, but now renamed International Article Number even though the abbreviation EAN has been retained) is a 13 digit (12 data and 1 check) barcoding standard which is a superset of the original 12-digit Universal Product Code (UPC) system. The EAN-13 barcode is defined by the standards organization GS1.

The 13 digits in the EAN-13 barcode are grouped as follows:

- The left group: Digits 2-7. The left group also encodes digit 1, through a scheme of odd and even parity.
- The right group: Digits 8-13, digit 13 is the check digit. -Wikipedia.

13.6.2 ERP

Enterprise resource planning (ERP) is business management software—typically a suite of integrated applications—that a company can use to collect, store, manage and interpret data from many business activities.

13.7 G

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

13.7.1 GS1 Standards

Most companies initially come to GS1 to get a bar code number for their products. However, GS1 standards provide a much wider framework for supply chain visibility. The current architecture of GS1 standards is as follows:

- Identify: Standards for the identification of items, locations, shipments, assets, etc.. and associated data.
- Capture: Standards for encoding and capturing data in physical data carriers such as barcodes and RFID tags.
- Share: Standards for sharing data between parties. -Wikipedia.

13.7.2 GTIN

Global Trade Item Number (GTIN) is an identifier for trade items developed by GS1 (comprising among others of the former EAN International and Uniform Code Council). Such identifiers are used to look up product information in a database (often by entering the number through a bar code scanner pointed at an actual product) which may belong to a retailer, manufacturer, collector, researcher, or other entity.

GTINs may be 8, 12, 13 or 14 digits long, and each of these 4 numbering structures are constructed in a similar fashion, combining Company Prefix, Item Reference and a calculated Check Digit (GTIN-14 adds another component - the Indicator Digit, which can be 1-8). GTIN-8s will be encoded in an EAN-8 bar code. GTIN-12s may be shown in UPC-A, ITF-14, or GS1-128 bar codes. GTIN-13s may be encoded in EAN-13 , ITF-14 or GS1-128 bar codes, and GTIN-14s may be encoded in ITF-14 or GS1-128 bar codes. The choice of bar code will depend on the application; for example, items to be sold at a retail establishment should be marked with EAN-8 , EAN-13 , UPC-A or UPC-E bar codes. -Wikipedia.

13.8 H

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

13.8.1 HMI

Human Machine Interface (HMI) is the user interface in a manufacturing or process control system. It provides a graphical display of an industrial control and monitoring system. An HMI typically resides in an office-based Windows computer that communicates with a specialized computer in the plant such as a programmable logic controller (PLC). HMI is now more widely used for industrial softwares (such as Ignition) that monitor and control entire plants and provide historical and statistical data. - PC Magazine Encyclopedia

13.9 I

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

13.9.1 Individual Control Charts

In statistical quality control, the individual/moving-range chart is a type of control chart used to monitor variables data from a business or industrial process for which it is impractical to use rational subgroups. The chart is necessary in the following situations:

1. Where automation allows inspection of each unit, so rational subgrouping has less benefit.
2. Where production is slow so that waiting for enough samples to make a rational subgroup unacceptably delays monitoring.
3. For processes that produce homogeneous batches (e.g., chemical) where repeat measurements vary primarily because of measurement error. -Wikipedia

13.10 J

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

13.11 K

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

13.12 L

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

13.12.1 LAN

A local area network (LAN) is a computer network that interconnects computers within a limited area. It shares a common communication line or wireless link to a server. -Wikipedia

13.12.2 LCL

Bottom limit in quality control for data points below the control (average) line in a control chart. Opposite of upper control limit. The horizontal line drawn on a control chart at a specified distance below the central line; points plotted below the lower control limit indicate that the process may be out of control.

13.12.3 LEAN

Lean manufacturing or lean production, often simply "lean", is a systematic method for the elimination of waste within a manufacturing system. Lean also takes into account waste created through overburden and waste created through unevenness in work loads. -Wikipedia

13.12.4 Lean Six Sigma

Lean Six Sigma is a methodology that relies on a collaborative team effort to improve performance by systematically removing waste; combining lean manufacturing/lean enterprise and Six Sigma to eliminate the eight kinds of waste.-Wikipedia

13.13 M

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

13.13.1 MES

A **manufacturing execution system** (**MES**) is a control system for managing and monitoring work-in-process on a factory floor.

13.13.2 MOM

Manufacturing operation management (MOM) is an approach of overseeing all aspects of the manufacturing process with a particular focus to increase efficiency.

13.14 N

13.14.1 Normal Distribution

In probability theory, the normal (or Gaussian) distribution is a very common continuous probability distribution. Normal distributions are used in the natural and social sciences to represent real-valued random variables whose distributions are not known. The normal distribution is remarkably useful because of the central limit theorem. In its most general form, under some conditions (which include finite variance), it states that averages of random variables independently drawn from independent distributions converge in distribution to the normal, that is, become normally distributed when the number of random variables is sufficiently large. Physical quantities that are expected to be the sum of many independent processes (such as measurement errors) often have distributions that are nearly normal. Moreover, many results and methods (such as propagation of uncertainty and least squares parameter fitting) can be derived analytically in explicit form when the relevant variables are normally distributed. The normal distribution is sometimes informally called the bell curve. However, many other distributions are bell-shaped (such as Cauchy's, Student's, and logistic). The terms Gaussian function and Gaussian bell curve are also ambiguous because they sometimes refer to multiples of the normal distribution that cannot be directly interpreted in terms of probabilities. The probability density of the normal distribution is:

$$F(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2 / 2\sigma^2}$$

Here, is the mean or expectation of the distribution (and also its median and mode). The parameter is its standard deviation with its variance then ^{2}. A random variable with a Gaussian distribution is said to be normally distributed and is called a normal deviate. If = 0 and = 1, the distribution is called the standard normal distribution or the unit normal distribution denoted by N (0,1) and a random variable with that distribution is a standard normal deviate. -Wikipedia.

13.15 O

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

13.15.1 OEE

OEE (Overall Equipment Effectiveness) measures the percentage of planned production time that is truly productive. OEE = Availability X Performance X Quality.

13.15.2 OIT

OIT (Operator Interface Terminal) is a low level graphical interface to a specified computer on the plant floor such as a programmable automation controller (PAC), programmable logic controller (PLC) or distributed control system (DCS).

13.15.3 OPC Server

An OPC Server is a software application that acts as an API (Application Programming Interface) or protocol converter. An OPC Server can connect to a device such as a PLC, DCS, and RTU or a data source such as a database or HMI to translate the data into a standard OPC format.

13.15.4 OPC-UA

OPC Unified **A**rchitecture (**OPC-UA**) is a communication protocol for interoperability developed by the OPC Foundation . The Foundation's goal for this project was to provide a path forward from the original OPC communications model (namely COM/DCOM) to a cross-platform service-oriented architecture (SOA) for process control, while enhancing security and providing an information model.

Using the OPC-UA module, Ignition acts as an OPC-UA server, serving data collected by its built in drivers to other Ignition modules, as well as third-party OPC-UA clients. -Wikipedia

13.16 P

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

13.16.1 PLC

Programmable **L**ogic **C**ontroller (**PLC**) is a programmable microprocessor-based device that is used in manufacturing to control assembly lines and machinery on the shop floor as well as many other types of mechanical, electrical, and electronic equipment in a plant. A PLC is designed for real-time use in rugged, industrial environments. Connected to sensors and actuators, PLCs are categorized by the number and type of I/O ports they provide and by their I /O scan rate.

13.16.2 PLM

Product lifecycle management (PLM) is an information management system that can integrate data, processes, business systems and, ultimately, people in an extended enterprise.

13.16.3 PPM

PPM (Parts per million) is a measurement used today by many customers to measure quality performance. Definition: One PPM means one (defect or event) in a million or 1/1,000,000.

13.16.4 Python dictionary

A Python dictionary is a key-value store container. A dictionary allows the developer to assign an index or “key” for every element that is placed into the construct. Therefore, each entry into a dictionary requires two values, the key and the element.

13.17 Q

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

13.18 R

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

13.18.1 Regex

In theoretical computer science and formal language theory, a regular expression (sometimes called a rational expression or regex) is a sequence of characters that define a search pattern, mainly for use in pattern matching with strings, or string matching, i.e. "find and replace"-like operations and came into common use with the Unix text processing utilities ed, an editor, and grep, a filter. -Wikipedia.

13.18.2 REST

In computing, representational state transfer (REST) is the software architectural style of the World Wide Web. REST gives a coordinated set of constraints to the design of components in a distributed hypermedia system that can lead to a higher-performing and more maintainable architecture.

The architectural properties affected by the constraints of the REST architectural style are:

- Performance - component interactions can be the dominant factor in user-perceived performance and network efficiency.
- Scalability - support large numbers of components and interactions among components. -Wikipedia.

13.18.3 RS-232

In telecommunications, RS-232 is a standard for serial communication transmission of data. It formally defines the signals connecting between a DTE (data terminal equipment) such as a computer terminal, and a DCE (data circuit-terminating equipment), such as a modem. The RS-232 standard is commonly used in computer serial ports. The standard defines the electrical characteristics and timing of signals, the meaning of signals, and the physical size and pinout of connectors. -Wikipedia.

13.19 S

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

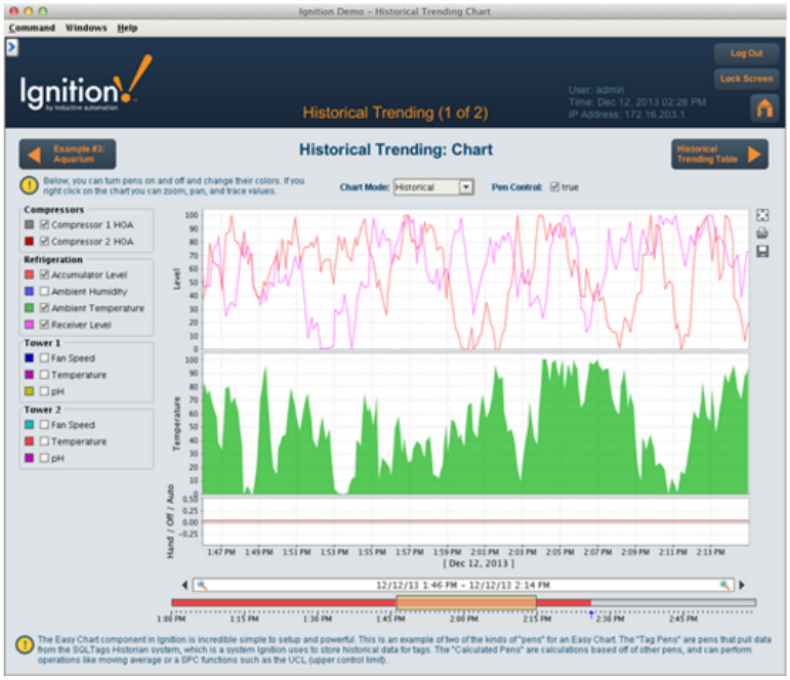
13.19.1 SCADA

Supervisory **C**ontrol **A**nd **D**ata **A**cquisition (**SCADA**) is a process control system that is used in countless number of applications, including manufacturing, communications, distribution (water, gas, power) and heating, cooling and security in buildings. A SCADA system collects data from sensors in local and remote locations and sends them to central computers to control local machinery.

SCADA systems range from simple configurations to large and complex projects. Most SCADA systems use HMI (human-machine interface) software that allows users to interact with and control the machines and devices that the HMI is connected to such as valves, pumps, motors, and much more.

SCADA software receives its information from **RTU**s (**R**emote **T**erminal **U**nits) or **PLC**s (**P**rogrammable **L**ogic **C**ontrollers) which can receive their information from sensors or manually inputted values. From here, the data can be used to effectively monitor, collect and analyze data, which can potentially reduce waste and improve efficiency resulting in savings of both time and money.

Ignition is a SCADA software solution



13.19.2 Six Sigma

Six Sigma is a set of techniques and tools for process improvement. It seeks to improve the quality of the output of a process by identifying and removing the causes of defects and minimizing variability in manufacturing and business processes. It uses a set of quality

managementmethods, mainly empirical, statistical methods, and creates a special infrastructure of people within the organization, who are experts in these methods. Each Six Sigma project carried out within an organization follows a defined sequence of steps and has specific value targets, for example: reduce process cycle time, reduce pollution, reduce costs, increase customer satisfaction, and increase profits. -Wikipedia

13.19.3 SOA

A service-oriented architecture (SOA) is an architectural pattern in computer software design in which application components provide services to other components via a communications protocol, typically over a network. The principles of service-orientation are independent of any vendor, product or technology. A service is a self-contained unit of functionality, such as retrieving an online bank statement. By that definition, a service is a discretely invokable operation. -Wikipedia

13.19.4 SPC

Statistical process control (SPC) is a method of quality control which uses statistical methods . SPC is applied in order to monitor and control a process. -Wikipedia

13.19.5 SUDS

In computer science, a succinct data structure (SUDS) is a data structure which uses an amount of space that is "close" to the information-theoretic lower bound, but (unlike other compressed representations) still allows for efficient query operations. The concept was originally introduced by Jacobson to encode bit vectors, (unlabeled) trees, and planar graphs. Unlike general lossless data compression algorithms, succinct data structures retain the ability to use them in-place, without decompressing them first. A related notion is that of a compressed data structure, in which the size of the data structure depends upon the particular data being represented. -Wikipedia

13.20 T

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

13.20.1 TEEP

TEEP stands for Total Effective Equipment Performance. TEEP is a performance metric that shows the total performance of equipment based on the amount of time the equipment was present. Typically the equipment is onsite and thus TEEP is metric that shows how well equipment is utilized.

TEEP = Planned Equipment Downtime% X Availability% X Performance% X Quality%

13.21 U

13.21.1 UCL

A value that indicates the highest level of quality acceptable for a product or service. The upper control limit is used in conjunction with the lower control limit to create the range of variability for quality specifications, enabling those within the organization to provide an optimal level of excellence by adhering to the established guidelines.

13.21.2 UPC

The Universal Product Code (UPC) is a barcode symbology (i.e., a specific type of barcode) that is used for scanning of trade items at the point of sale, per GS1 specifications. The most common form, UPC-A, consists of 12 numerical digits, which are uniquely assigned to each trade item. -Wikipedia.

13.22 V

13.22.1 VPN

A virtual private network (VPN) extends a private network across a public network, such as the Internet. It enables users to send and receive data across shared or public networks as if their computing devices were directly connected to the private network, and thus are benefiting from the functionality, security and management policies of the private network. -Wikipedia

13.23 W

13.24 X

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

13.24.1 XBarS chart

In statistical quality control, the x bar and s chart is a type of control chart used to monitor variables data when samples are collected at regular intervals from a business or industrial process. An Xbar-S chart plots the process mean (Xbar chart) and process standard deviation (S chart) over time for variables data in subgroups. -Wikipedia.

13.25 Y

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

13.26 Z

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z