Point85
Overall Equipment Effectiveness Applications
Version 1.0

Kent Randall
July 10, 2018

Table of Contents

Overview 3
OEE Calculations 4
Technologies 5
Installation 5
Designer Application 5
Plant Entity Editor 6
Entity Hierarchy 6
Equipment Processed Materials 7
Equipment Data Collection Events 8
Material Editor 10
Reason Editor 11
Unit of Measure (UOM) 13
Description 13
Editor 14
Converter 17
Work Schedule 17
Description 17
Editor 18
Shifts 19
Rotations 20
Teams 21
Non-working Periods 21
Import Schedule 22
Scripting 23
Description 23
Editor 26
Data Source Editors 27
OPC UA Data Source 27
Browser 27
Trending 29
OPC DA Data Source 31
Browser 31
Trending 33
HTTP Data Source 33
Definition 33
Java Client Example 34
Database Trigger Example 35
iOS/Android Example 37
Trending 38
Messaging Data Source 39

Definition 39
Trending 40
Dashboard 42
Collector Application 42
Monitor Application 42
Operator Application 42
Testing Applications 44
Collector User Interface 44
HTTP and Messaging 45
HTTP 46
Messaging 47
Database 47
Design Schema 47
PLANT_ENTITY Table 47
MATERIAL Table 47
REASON Table 47
Execution Schema 48
Contributing Technology 48
References 49

OVERVIEW

The Point85 Overall Equipment Effectiveness (OEE) applications enable:

- collection of equipment data from multiple sources to support OEE calculations ,
- resolution of a collected data value into an availability reason or produced material quantity to provide input to the performance, availability and quality components of OEE
- calculation of the OEE key performance indicator (KPI) for the equipment using an optional work schedule for defining the scheduled production time
- monitoring of equipment availability, performance and quality events

Sources of equipment availability, performance and quality event data include:

- *Manual*: web browser-based data entry
- *OPC DA*: classic OLE for Process Control (OPC) Data Acquisition (DA)
- *OPC UA*: OLE for Process Control Unified Architecture (UA)
- *HTTP*: invocation of a web service via an HTTP request
- *Messaging*: an equipment event message received via a RabbitMQ message broker

The Point85 applications supporting OEE are:

- *Designer*: a GUI application for defining the plant equipment, data sources, event resolution scripts, manufacturing work schedule, availability reasons, produced materials and units of measure for data collectors. The designer also includes a dashboard and trending capabilities.
- *Collector*: a Windows service or Unix deamon to collect the equipment event data and store it in a relational database
- *Monitor*: a GUI application with a dashboard to view the current equipment OEE and status
- *Operator*: a web-application for manual entry of equipment events

In addition, two test applications assist in the development of an OEE solution:

- HTTP requester and RabbitMQ message publisher
- GUI front end for a data collector

OEE Calculations

OEE is the product of equipment availability, performance and quality each expressed as a percentage. The time-loss model is used to accumulate time in loss categories (or "no loss" if the equipment is running normally). See [Kennedy] for details. A data source provides an input value to a data collector's resolver JavaScript function that maps that input value to an output value (reason or production count).

For availability and performance, the output value is a reason that is assigned to one of the following loss categories:

- *Value Adding:* the "no loss" or "running OK" category.
- *Not Scheduled:* this is non-working time. Non-working periods (e.g. holidays) typically are planned in the work schedule that is assigned to a plant entity.
- *Unscheduled*: working time when the equipment is not scheduled for normal production (e.g. an R&D or laboratory test run).
- *Planned Downtime*: working time when the equipment is not scheduled for normal production but the activity is intended to support production (e.g. planned preventive maintenance).
- *Unplanned Downtime*: working time when the equipment is not available due to an unexpected fault (e.g. motor failure or jam).
- *Setup*: working time when the equipment is being changed over in order to run new material.
- *Stoppages*: minor or short periods of time when the equipment is not producing as expected (such as a blocked or starved condition).
- *Reduced Speed*: the equipment is producing, but not at its design speed or ideal run rate.

For quality or yield, the data source provides a production count in the good, reject/rework or startup & yield categories in the defined units of measure for the material being produced.

TECHNOLOGIES

Technologies used for the OEE applications are:

- Java 8 programing language and JavaFX for the GUIs
- JavaScript for resolving an input value into an output value based on an OEE event. The Nashorn script engine included in the Java 8 JVM is used.
- Vaadin and Tomcat for the web application
- j-Interop and OpenSCADA utgard for the OPC DA client
- Eclipse Milo for the OPC UA client
- Hibernate with a Hikari connection pool and JPA interface for persisting data to a relational database
- NanoHTTPD for the embedded HTTP server
- RabbitMQ and Erlang for the messaging middle-ware
- Google Gson for message and HTTP request serialization and deserialization

INSTALLATION

The applications are packaged in a zip file.

TBD

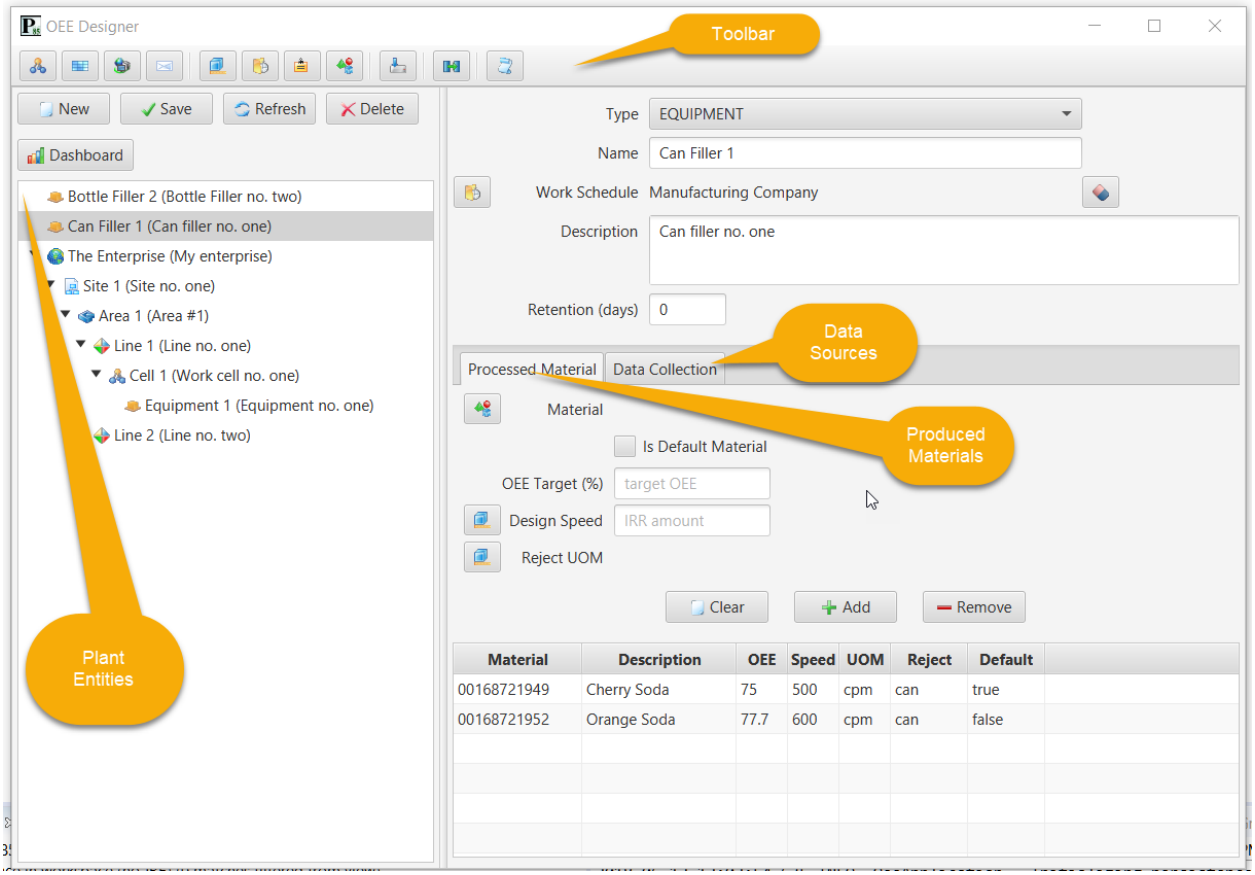
DESIGNER APPLICATION

The Designer application is used to define all aspects of an OEE solution. It is launched from a shell script. A splash screen is displayed during the time the database connection is being established and plant entity objects are being created.

The Designer is focused on configuring all aspects of a piece of equipment to enable its OEE calculations.

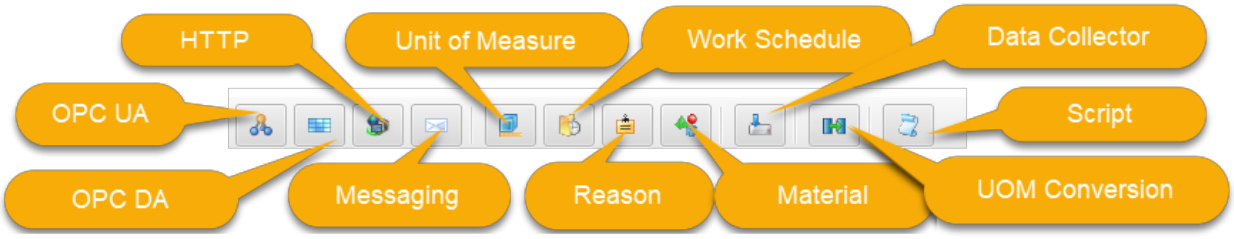
Plant Entity Editor

Upon launch, the plant entity editor is displayed, for example:



Toolbar

The toolbar buttons that launch other editors or applications (as discussed below) are:



Entity Hierarchy

To begin, the plant entity physical model on the left would typically be defined first. This is the ISA-95 organizational hierarchy of Enterprise -> Site -> Area -> Production Line -> Work Cell -> Equipment. Only equipment can have OEE calculations, but higher level entities can have an associated work schedule and data retention period that apply to equipment contained below them.

The physical model editor buttons are:

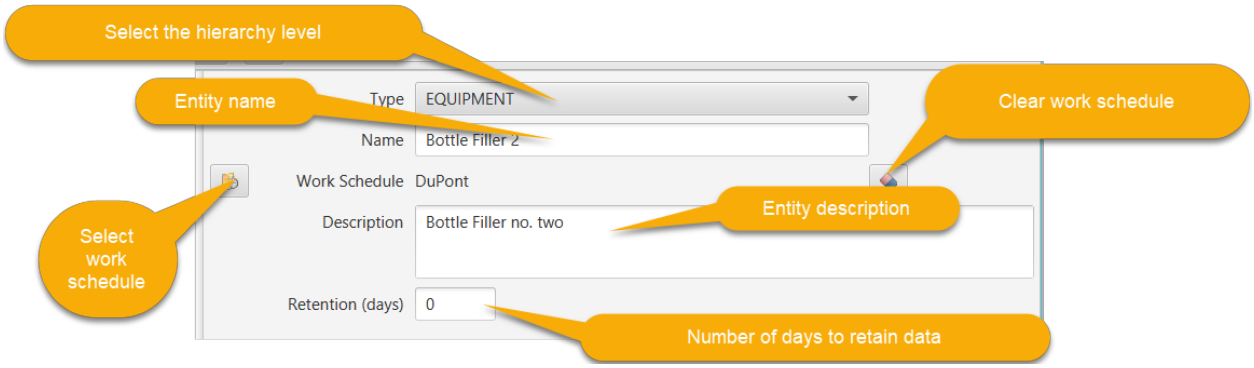
- *New*: clear the editor to begin defining a new entity
- *Save*: save the selected entity to the database. The parent entity (if any) must be selected first. If a parent is selected, the child must be created of the proper type (e.g. Equipment if the parent is a Work Cell).
- *Refresh*: refresh the selected entity from the database to synchronize the editor with the entity's saved state
- *Delete*: delete the selected entity and any children from the database

The Dashboard button displays the OEE dashboard for the selected equipment entity. The dashboard is discussed below.

The physical model has a context menu accessed by right-clicking in the left-hand pane. The menu items are:

- *Save All Entities*: save all entities in the hierarchy to the database
- *Refresh All Entities*: restore all entities in the hierarchy from their state in the database
- *Clear Selected Entity*: de-select the entity so that no entity is selected

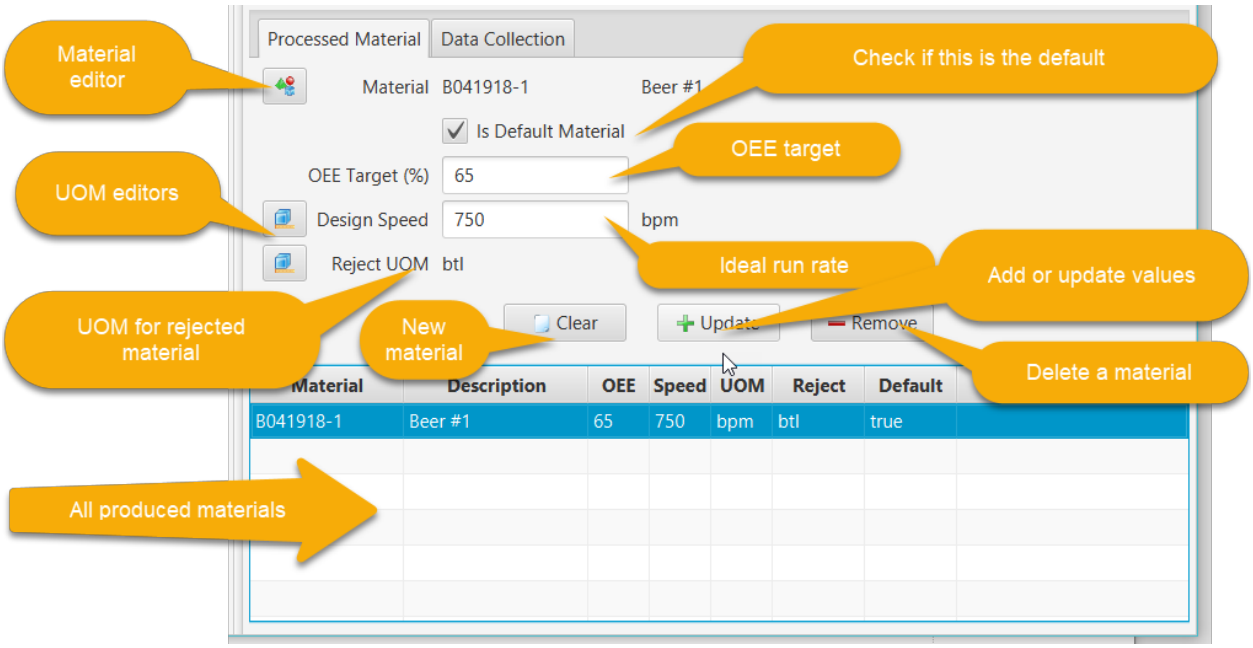
The entity's attributes are displayed and edited in the upper right-hand corner of the editor:



The work schedule and data retention days apply to that entity and to all children below it. For example, a work schedule could be defined for each Site or Area within a plant and have it apply to the contained equipment. A retention of 90 days means that any event records older than 90 days for that equipment will be deleted from the database when a new event is recorded.

Equipment Processed Materials

A piece of equipment can produce many different materials. This one-to-many relationship is defined in the Processed Material tab:



The editor actions are:

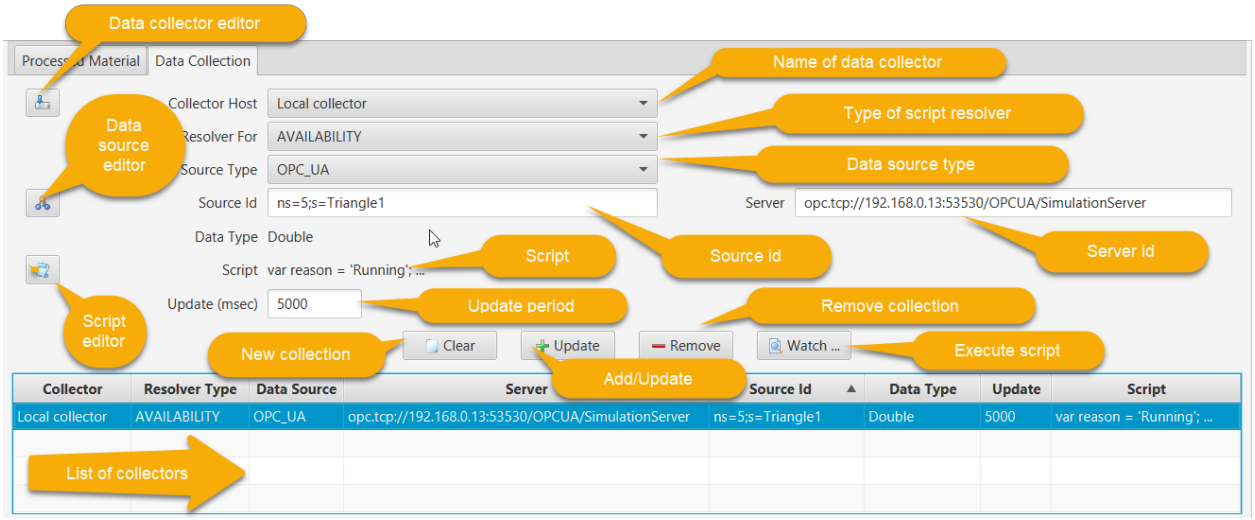
- *Clear*: clear the editor controls in order to define a new processed material. The Add/Update button text will change to "Add".
- *Add*: After defining the properties of a new material, clicking this button will add it to the list of materials
- *Update*: after selecting an existing processed material, the current values will be moved into the editing controls. Make the necessary edits, then click this button to apply the changes to the list.
- *Remove*: after selecting an existing processed material, click this button to remove it from the list.

Note that after making changes to the list of processed materials, the plant entity must be saved to the database by clicking the Save button. An unsaved entity will be marked as such.

The material editor is accessed by clicking on the material button in this tab. The produced material is then selected in the editor and the dialog closed. The unit of measure editor for the design speed and rejects is accessed in a similar fashion.

Equipment Data Collection Events

A piece of equipment can many different sources of availability and production events. This one-to-many relationship is defined in the Data Collection tab:



The editor actions are:

- *Clear*: clear the editor controls in order to define a new resolver. The Add/Update button text will change to "Add".
- Add: After defining the properties of the resolver, clicking this button will add it to the list
- *Update*: after selecting an existing resolver, the current values will be moved into the editing controls. Make the necessary edits, then click this button to apply the changes to the list.
- *Remove*: after selecting an existing resolver, click this button to remove it from the list.
- *Watch*: after selecting a resolver, click this button to launch a dialog to observe execution of the resolver script when a new input value arrives. The output resolution is not saved to the database. The dialogs are discussed below under the individual data source editors.

Note that after making changes to the list of event resolvers, the plant entity can be saved to the database by clicking the Save button.

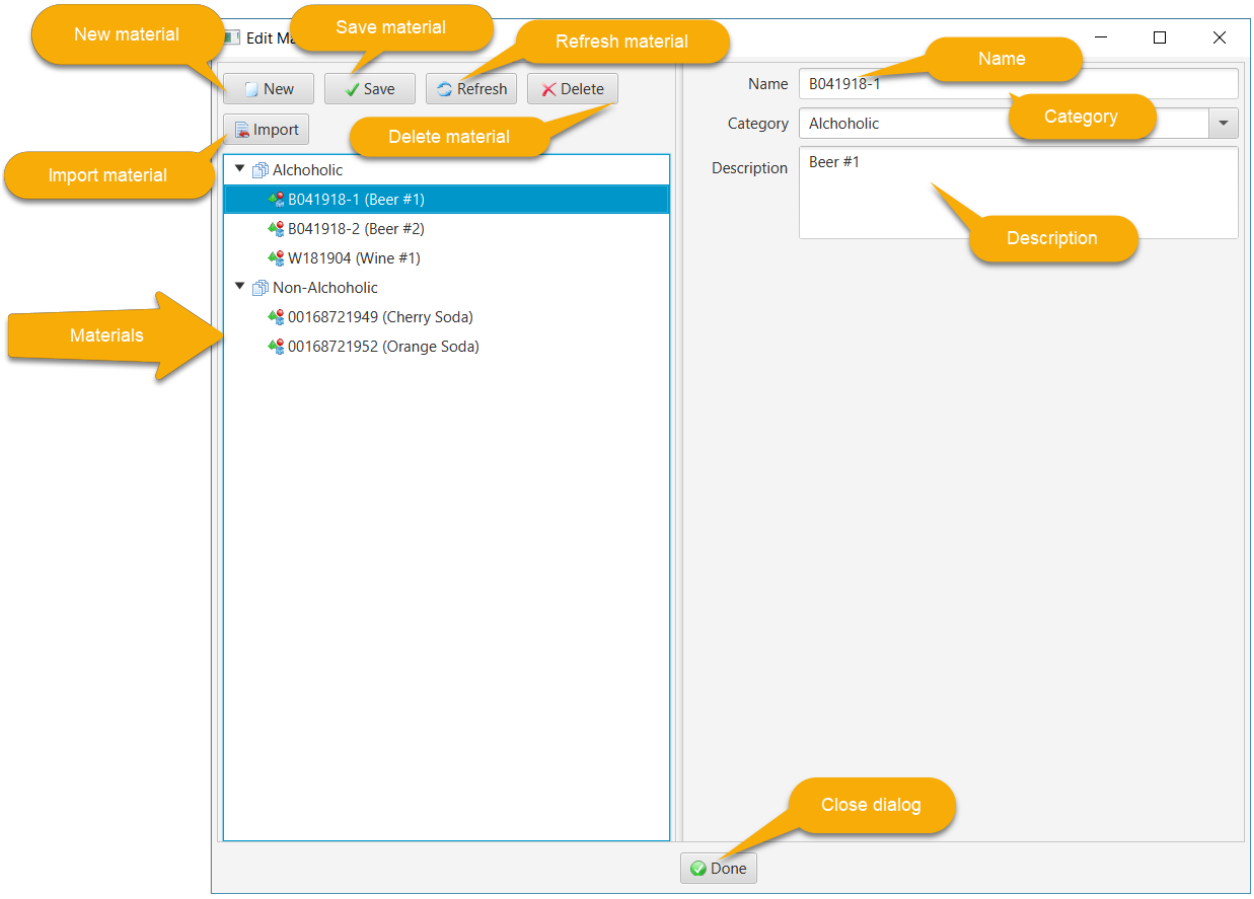
A data collector host is a Windows process or Unix daemon that interfaces to one or more data sources to collect data for OEE calculations. A previously defined data collector for this event can be selected in the dropdown, or the editor can be launched by clicking the editor button. This editor is discussed below.

The resolver type (availability, good production, reject/rework production, setup & yield production, material change or job/work order change) is selected in the next dropdown. The data source type (OPC DA, OPC UA, HTTP or messaging) is selected in the next dropdown.

The data source editor is launched by clicking on the button next to the source id field. For an OPC DA source, a tag is selected in the browser. For OPC UA a node is selected. For HTTP, the HTTP data collector's embedded server is selected (in this case, the source id will be automatically created). For a messaging source, the RabbitMQ broker is selected (in this case too, the source id will be automatically created).

MATERIAL EDITOR

Materials produced by equipment are defined in this editor. For example:



The material editor buttons are:

- *New*: clear the editor to begin defining a new material
- *Save*: save the selected material to the database.
- *Refresh*: refresh the selected material from the database to synchronize the editor with the material's saved state
- *Delete*: delete the selected material from the database
- *Import*: Import materials from a comma-separated value (CSV) file

The material editor has a context menu accessed by right-clicking in the left-hand pane. The menu items are:

- *Save All Material*: save all materials to the database
- *Refresh All Material*: restore all materials from their state in the database

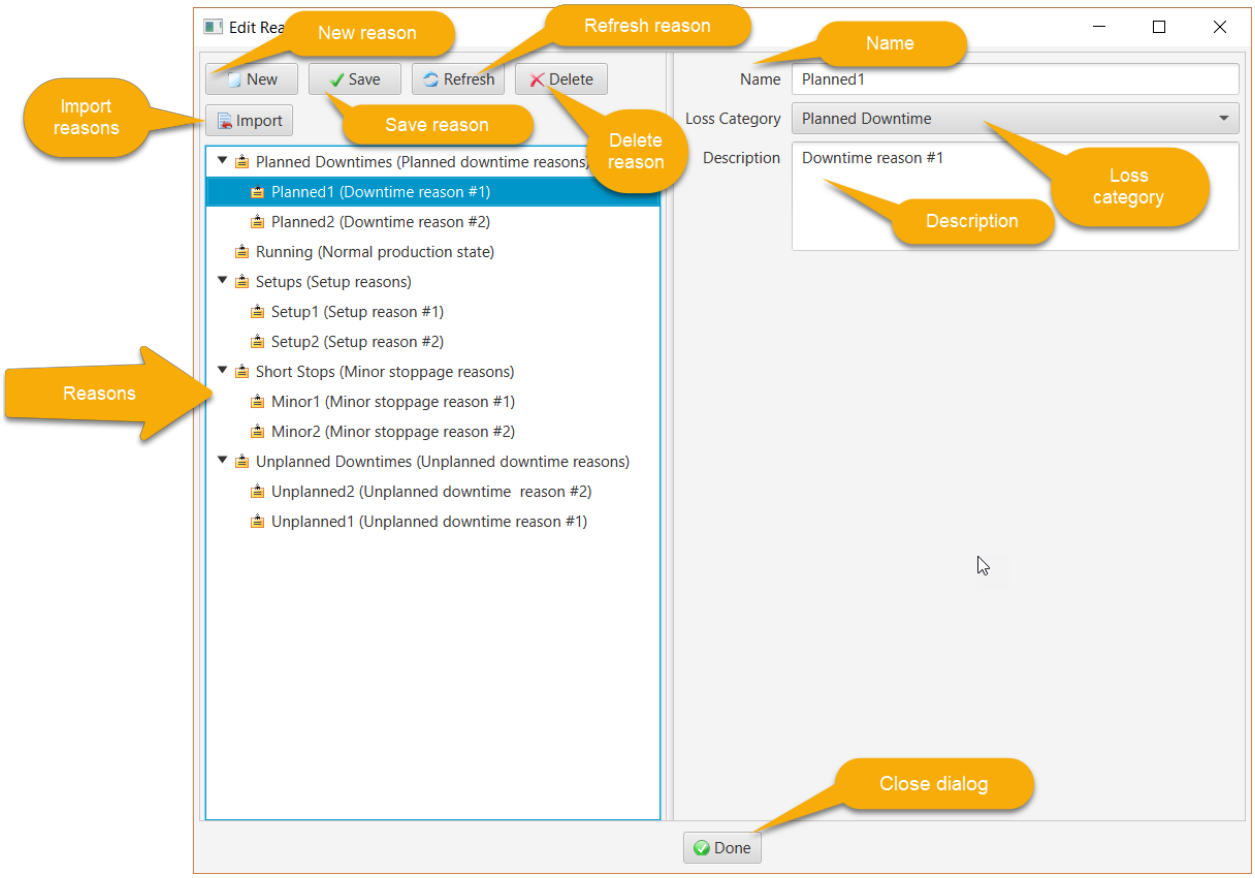
The material category provides a convenient way to organize different material.

Clicking the Import button launches a dialog to choose a text file with a comma-separated lines for each material. The format is “name, description, category”. The “materials.csv” file is included in the project as a example. For example:

```
B041918-1, Beer #1, Alcoholic

B041918-2, Beer #2, Alcoholic

W181904, Wine #1, Alcoholic

00168721952, Orange Soda, Non-Alcoholic

00168721949, Cherry Soda, Non-Alcoholic
```

REASON EDITOR

Availability and performance reasons are defined in this editor. For example:



The reason editor buttons are:

- *New*: clear the editor to begin defining a new reason
- *Save*: save the selected reason to the database.

- *Refresh*: refresh the selected reason from the database to synchronize the editor with the reason's saved state
- *Delete*: delete the selected reason from the database
- *Import*: Import reasons from a file

When saving a reason, if a parent reason is selected after creating the new reason, you will be asked to confirm that the new reason is a child of this parent. This hierarchy provides a way to organize the reasons. If a reason is used to determine an availability or performance event, it must have a loss category. Any reason with a loss category can be used in an OEE availability event.

The reason editor has a context menu accessed by right-clicking in the left-hand pane. The menu items are:

- *Save All Reasons*: save all reasons to the database
- *Refresh All Reasons*: restore all reasons from their state in the database
- *Clear Selected Reason*: unselect the selected reason so that no reason is selected

Clicking the Import button launches a dialog to choose a text file with comma-separated lines for each reason. The "reasons.csv" file is included in the project as a example. The format is "name, description, loss category, parent reason". For example:

```
Running, Normal production state, NO_LOSS
Setups, Setup reasons, ,
Setup1, Setup reason #1, SETUP, Setups
Setup2, Setup reason #2, SETUP, Setups
Planned Downtimes, Planned downtime reasons, ,
Planned1, Downtime reason #1, PLANNED_DOWNTIME, Planned Downtimes
Planned2, Downtime reason #2, PLANNED_DOWNTIME, Planned Downtimes
Unplanned Downtimes, Unplanned downtime reasons, ,
Unplanned1, Unplanned downtime reason #1, UNPLANNED_DOWNTIME, Unplanned Downtimes
Unplanned2, Unplanned downtime  reason #2, UNPLANNED_DOWNTIME, Unplanned Downtimes
Short Stops, Minor stoppage reasons, ,
Minor1, Minor stoppage reason #1, MINOR_STOPPAGES, Short Stops
Minor2, Minor stoppage reason #2, MINOR_STOPPAGES, Short Stops
```

The loss category name must match the TimeLoss.java enum name:

- `NO_LOSS`
- `NOT_SCHEDULED`
- `UNSCHEDULED`
- `PLANNED_DOWNTIME`
- `SETUP`

- UNPLANNED_DOWNTIME
- MINOR_STOPPAGES
- REDUCED_SPEED
- REJECT_REWORK
- STARTUP_YIELD

UNIT OF MEASURE (UOM)

Description

Good, reject/rework and setup/yield production quantities must have a unit of measure. The equipment's design speed (a.k.a. ideal run rate) must be a quotient UOM, i.e. rate. The reject/rework units must also be specified. These units of measure are used in OEE calculations when the input and output UOMs differ. For example, a case packer might accept "can" as the input and reject UOM, but output a "case" of 12 cans.

A measurement system is a collection of units of measure where each pair has a linear relationship, i.e. y = ax + b where 'x' is the abscissa unit to be converted, 'y' (the ordinate) is the converted unit, 'a' is the scaling factor and 'b' is the offset. In the absence of a defined conversion, a unit will always have a conversion to itself where a = 1 and b = 0. A bridge unit conversion is defined to convert between the fundamental SI and International customary units of mass (i.e. kilogram to pound mass), length (i.e. metre to foot) and temperature (i.e. Kelvin to Rankine). These three bridge conversions permit unit of measure conversions between the two systems. A custom unit can define any bridge conversion such as a bottle to US fluid ounces or litres if needed.

A simple unit, for example a metre, is defined as a scalar UOM. A special scalar unit of measure is unity or dimensionless "1".

A unit of measure that is the product of two other units is defined as a product UOM. An example is a Joule which is a Newton·metre.

A unit of measure that is the quotient of two other units is defined as a quotient UOM. An example is a velocity, e.g. metre/second.

A unit of measure that has an exponent on a base unit is defined as a power UOM. An example is area in metre^2. Note that an exponent of 0 is unity, and an exponent of 1 is the base (root) unit itself. An exponent of 2 is a product unit where the multiplier and multiplicand are the base unit. A power of -1 is a quotient unit of measure where the dividend is 1 and the divisor is the base unit.

Units are classified by type, e.g. length, mass, time and temperature. Only units of the same type can be converted to one another. System pre-defined units of measure are also enumerated, e.g. kilogram, Newton, metre, etc. Custom units (e.g. a 1 litre bottle) do not have a pre-defined type or enumeration and are referred to by a unique base symbol.

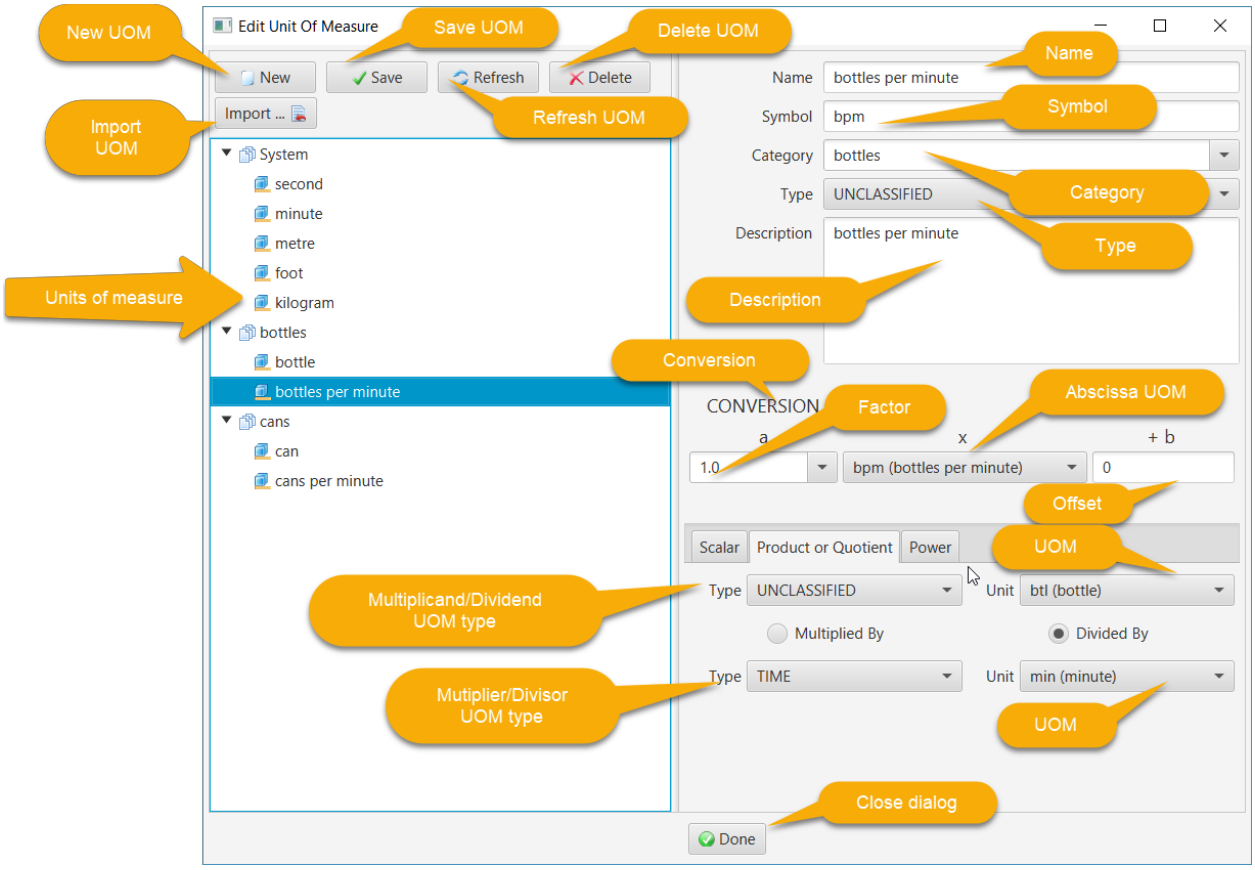
All units have a base symbol that is the most reduced form of the unit. For example, a Newton is kilogram·metre/second^2. The base symbol is used in the measurement system to register each unit

and to discern the result of arithmetic operations on quantities. For example, dividing a quantity of Newton·metres by a quantity of metres results in a quantity of Newtons.

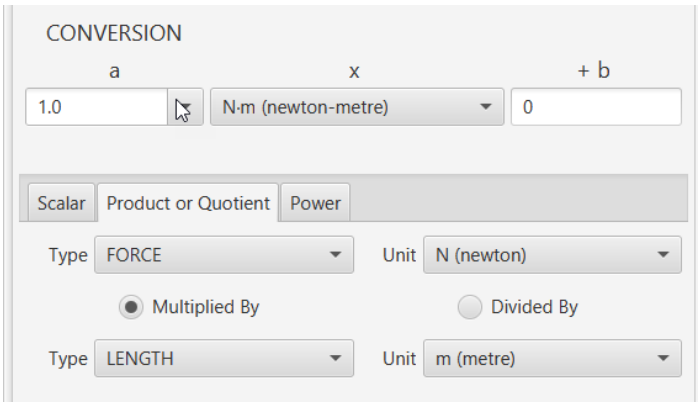
A quantity is an decimal amount together with a unit of measure. When arithmetic operations are performed on quantities, the original units can be transformed. For example, multiplying a length quantity in metres by a force quantity in Newtons results in a quantity of energy in Joules (or Newton-metres).

Editor

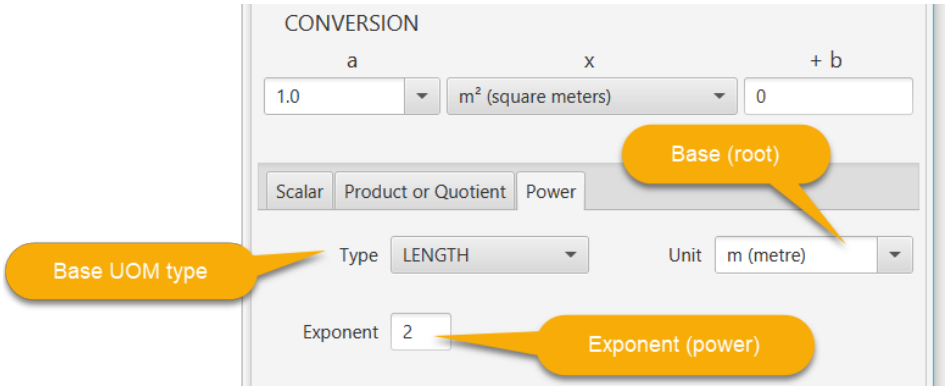
Units of measure are defined in the editor shown below. In this example, a quotient (rate) UOM "bottles per minute" has been created. The dividend is a scalar unit of "bottle" that was previously created. The divisor is a system time unit of "minute".



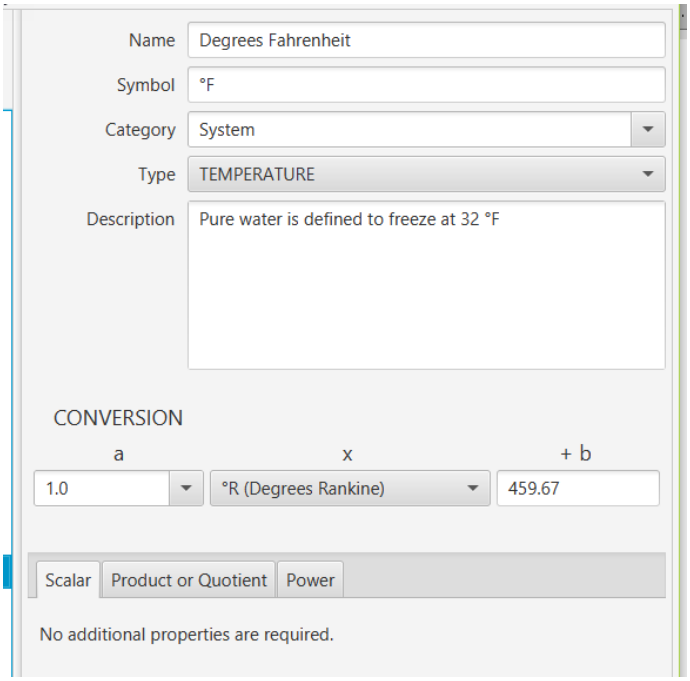
The UOM of measure editor is a general purpose editor and thus supports creation of quotient, product and power units as well as scalar ones (e.g. bottle, can, minute). For example, a Newton-metre is a system-defined product UOM created by importing it (as described below):



As a second example, square metres is a power UOM, again created by importing it:



Temperature in Fahrenheit is an example where the unit has a defined offset from Rankine units:



The UOM editor buttons are:

- *New*: clear the editor to begin defining a new UOM
- *Save*: save the selected UOM to the database.
- *Refresh*: refresh the selected UOM from the database to synchronize the editor with the UOM's saved state
- *Delete*: delete the selected UOM from the database
- *Import*: Import a system UOM from pre-defined choices

When creating a UOM, a category needs to be specified. The category provides a way to group related units. The "System" category is reserved for the pre-defined UOMs such as metre.

A UOM must also have a type. All units with the same type can be converted between each other. For packaging units like "can" or "bottle" the UNCLASSIFIED type should be chosen. Whether or not a conversion between units in this category makes sense is determined by application logic, since there is no physical basis for such units.

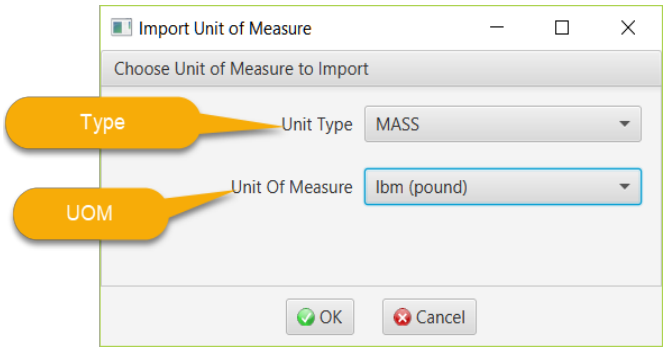
A large number of system-defined types and the corresponding UOMs are included:

- *dimension-less "1"*: UNITY
- *fundamental physical*: LENGTH, MASS, TIME, ELECTRIC_CURRENT, TEMPERATURE, SUBSTANCE_AMOUNT, LUMINOSITY
- *other physical*: AREA, VOLUME, DENSITY, VELOCITY, VOLUMETRIC_FLOW, MASS_FLOW, FREQUENCY, ACCELERATION, FORCE, PRESSURE, ENERGY, POWER, ELECTRIC_CHARGE, ELECTROMOTIVE_FORCE, ELECTRIC_RESISTANCE, ELECTRIC_CAPACITANCE, ELECTRIC_PERMITTIVITY, ELECTRIC_FIELD_STRENGTH, MAGNETIC_FLUX, MAGNETIC_FLUX_DENSITY, ELECTRIC_INDUCTANCE, ELECTRIC_CONDUCTANCE, LUMINOUS_FLUX, ILLUMINANCE, RADIATION_DOSE_ABSORBED, RADIATION_DOSE_EFFECTIVE, RADIATION_DOSE_RATE, RADIOACTIVITY, CATALYTIC_ACTIVITY, DYNAMIC_VISCOSITY, KINEMATIC_VISCOSITY, RECIPROCAL_LENGTH, PLANE_ANGLE, SOLID_ANGLE, INTENSITY, TIME_SQUARED, MOLAR_CONCENTRATION, IRRADIANCE
- *computer science*
- *currency*

The UOM editor has a context menu accessed by right-clicking in the left-hand pane. The menu items are:

- *Save All Units of Measure*: save all UOMs to the database
- *Refresh All Units of Measure*: restore all UOMs from their state in the database

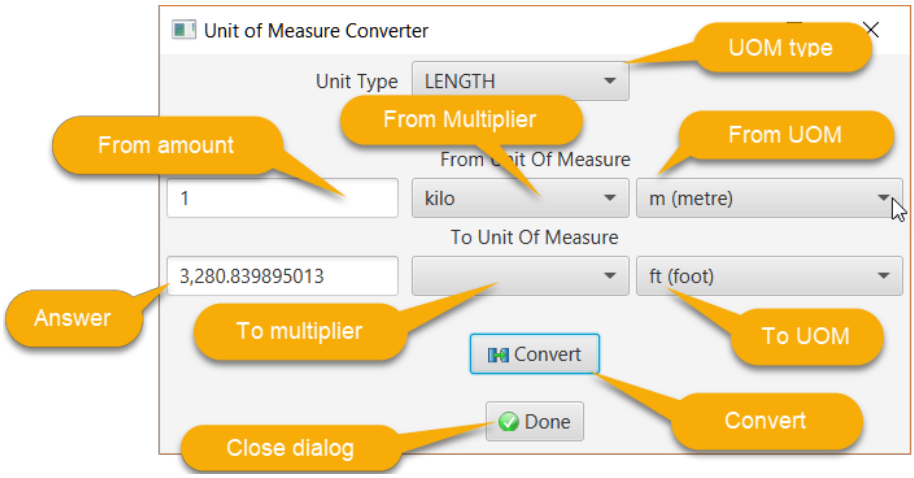
Clicking the Import button launches a dialog to choose the system-defined type and then the UOM of interest. For example to import the customary UOM of pound-mass:



Click "OK" to import the unit or "Cancel" to cancel out. Note that importing a UOM will import that UOM and all referenced units.

Converter

This utility is launched from the toolbar. It is used to convert from one UOM to another UOM of the same type (e.g. length to length). For example, 1 kilometre (1000 metre) is 3,280.8 feet:



First, choose the unit type (e.g. LENGTH), the enter the "from" amount (1), "from" factor if desired (kilo) and "from" unit (metre). Then, select the "to" factor (if any) and "to" unit (foot). Click the Convert button to display the answer of 3,280.8 feet.

Work Schedule

Description

The time equipment is scheduled for production is defined by a work schedule. The work schedule is attached to the equipment itself or to any node in the hierarchy above it. The search starts at the equipment with the availability event and moves up the hierarchy until a schedule is found. Therefore a work schedule could be defined for area or site and apply to all equipment below it.

A work schedule consists of one or more teams who rotate through a sequence of shift and off-shift periods of time. Breaks during shifts can be defined as well as non-working periods of time (e.g. holidays and scheduled maintenance periods) that are applicable to the entire work schedule.

A work schedule is defined with a name and description. Zero or more non-working periods can be defined. A non-working period has a defined starting date and time of day and duration. For example, the New Year's Day holiday starting at midnight for 24 hours, or three consecutive days for preventive maintenance of manufacturing equipment starting at the end of the night shift.

A shift is defined with a name, description, starting time of day and duration. An off-shift period is associated with a shift. Shifts can be overlapped (typically when a hand-off of duties is important). A rotation is a sequence of shifts and off-shift days. An instance of a shift has a starting date and time of day and has an associated shift definition.

A team/crew is defined with a name and description. It has a rotation with a starting date. The starting date shift will have an instance with that date and a starting time of day as defined by the shift. The same rotation can be shared between more than one team, but with different starting times.

A rotation is a sequence of working periods (segments). Each segment starts with a shift and specifies the number of days on-shift and off-shift. A work schedule can have more than one rotation.

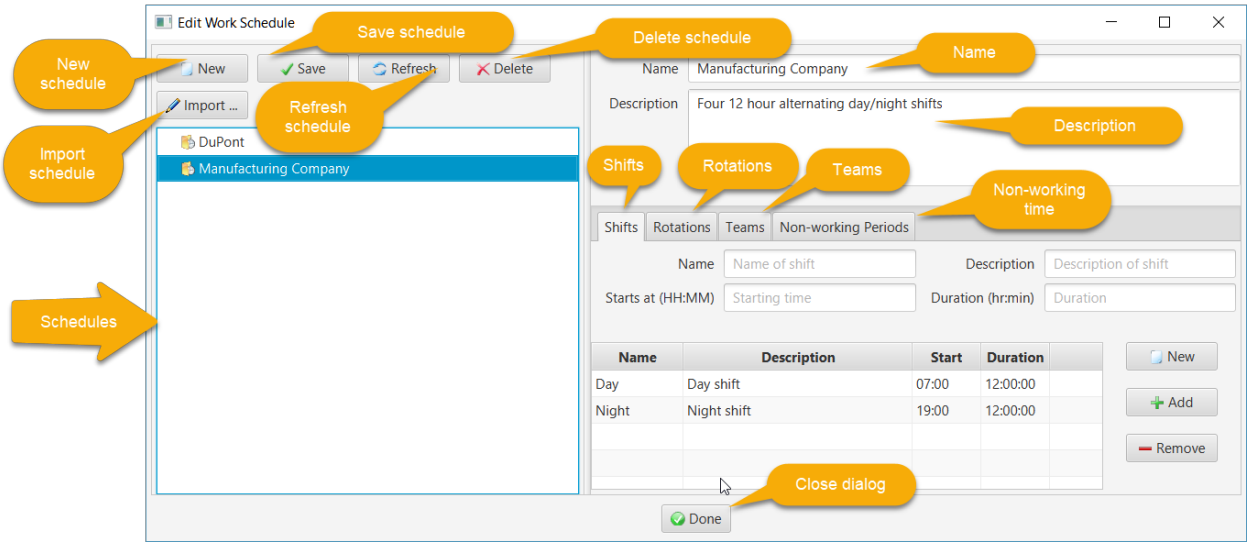
A non-working period is a duration of time where no production teams are working. For example, a holiday or a period of time when a plant is shutdown for preventative maintenance. A non-working period starts at a defined day and time of day and continues for the specified duration of time.

A shift instance is the duration of time from a specified date and time of day and continues for the duration of the associated shift. A team works this shift instance.

After a work schedule is defined, the working time for all shifts can be computed for a defined time interval. This duration of time is the maximum available productive time and is the input to the calculation of Overall Equipment Effectiveness (OEE). Time accumulated in the various loss categories subtracts from this total time to finally arrive at the value adding time. The shift when an OEE event occurs will be also recorded in the database.

Editor

For the work schedule editor shown below, the Manufacturing Company schedule has been selected:



The work schedule editor buttons are:

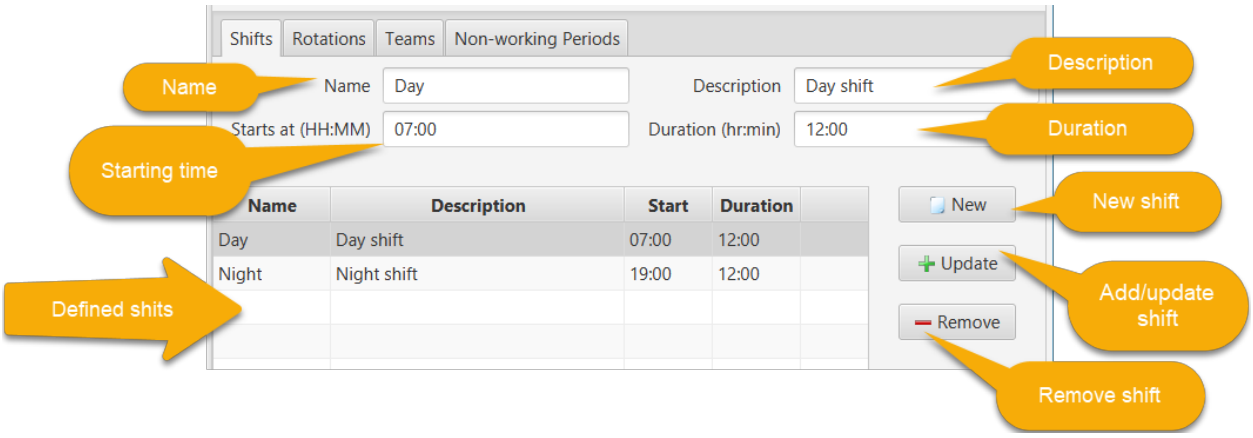
- *New*: clear the editor to begin defining a new schedule
- *Save*: save the selected schedule to the database.
- *Refresh*: refresh the selected schedule from the database to synchronize the editor with the schedule's saved state
- *Delete*: delete the selected schedule from the database
- *Import*: import a schedule from pre-defined templates

The work schedule editor has a context menu accessed by right-clicking in the left-hand pane. The menu items are:

- *Save All Schedules*: save all schedules to the database
- *Refresh All Schedules*: restore all schedules from their state in the database

Shifts

The "Shifts" tab is used to define shifts:



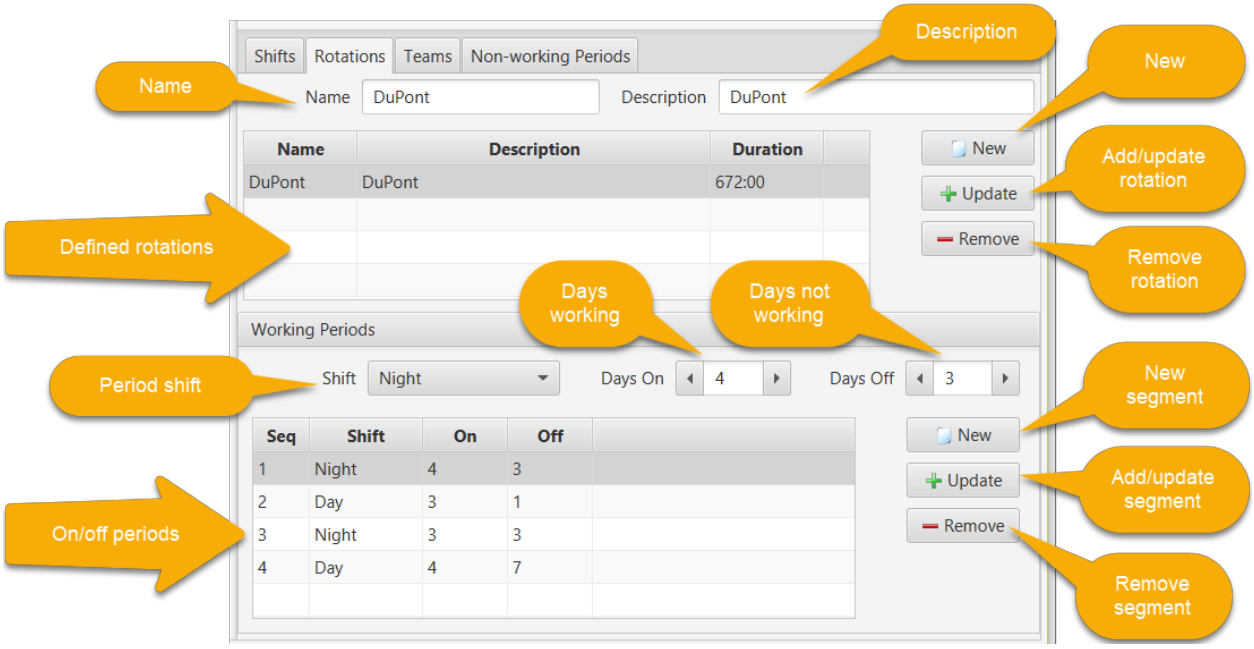
The editor actions are:

- *New*: clear the editor controls in order to define a new shift. The Add/Update button text will change to "Add".
- *Add*: After defining the properties of a new shift, clicking this button will add it to the list of shifts
- *Update*: after selecting an existing shift, the current values will be moved into the editing controls. Make the necessary edits, then click this button to apply the changes to the list.
- *Remove*: after selecting an existing shift, click this button to remove it from the list.

The starting time of day is entered in 24-hour format (hours:minutes from 00:00 to 23:59). The shift duration is entered as hours:minutes between 00:00 and 24:00. Note that after making changes to the list of shifts, the work schedule must be saved to the database by clicking the Save button.

Rotations

The "Rotations" tab is used to define rotations:



In this example, the DuPont 12-hour rotating shift schedule uses 4 teams (crews) and 2 twelve-hour shifts to provide 24/7 coverage. It consists of a 4-week cycle where each team works 4 consecutive night shifts, followed by 3 days off duty, works 3 consecutive day shifts, followed by 1 day off duty, works 3 consecutive night shifts, followed by 3 days off duty, work 4 consecutive day shift, then have 7 consecutive days off duty. Personnel works an average 42 hours per week.

This DuPont example has one long rotation of 672 hours (28 days or 4 weeks) consisting of 4 segments in this sequence:

1. Night shift, 4 days on followed by 3 days off
2. Day shift, 3 days on followed by 1 day off
3. Night shift, 3 days on followed by 3 days off
4. Day shift, 4 days on followed by 7 days off

The editor actions for rotations are:

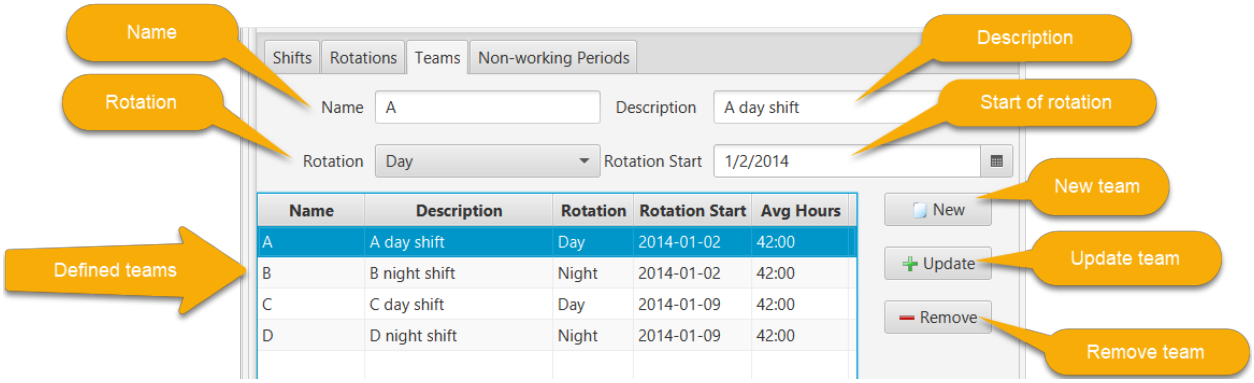
- *New*: clear the editor controls in order to define a new rotation. The Add/Update button text will change to "Add".
- *Add*: After defining the properties of a new rotation, clicking this button will add it to the list of rotations
- *Update*: after selecting an existing rotation, the current values will be moved into the editing controls. Make the necessary edits, then click this button to apply the changes to the list.
- *Remove*: after selecting an existing rotation, click this button to remove it from the list.

The editor actions for rotation segments are:

- *New*: clear the editor controls in order to define a new rotation segment. The Add/Update button text will change to "Add".
- *Add*: After defining the properties of a new rotation segment, clicking this button will add it to the list of rotation segments
- *Update*: after selecting an existing rotation segment, the current values will be moved into the editing controls. Make the necessary edits, then click this button to apply the changes to the list.
- *Remove*: after selecting an existing rotation segment, click this button to remove it from the list.

Teams

The "Teams" tab is used to define teams (crews):

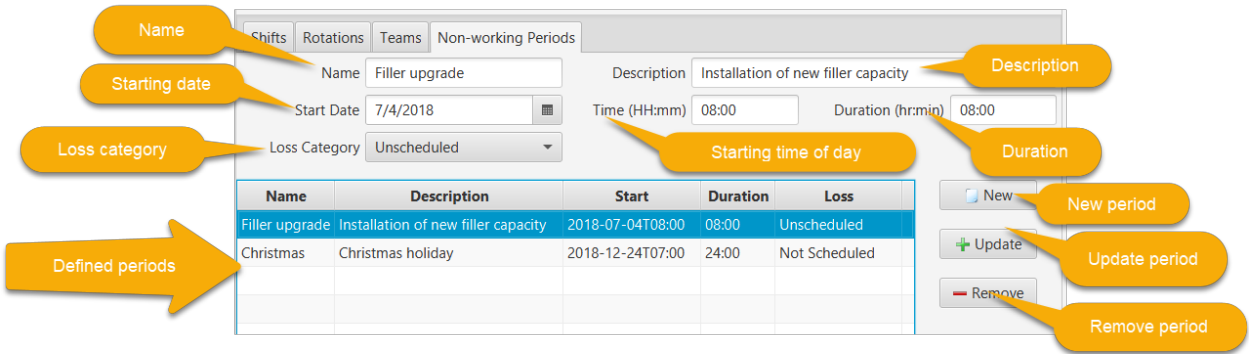


The editor actions for teams are:

- *New*: clear the editor controls in order to define a new team. The Add/Update button text will change to "Add".
- *Add*: After defining the properties of a new team, clicking this button will add it to the list of teams. The average working hours will be displayed in the last column, e.g. 42 hours and 0 minutes in this example.
- *Update*: after selecting an existing team, the current values will be moved into the editing controls. Make the necessary edits, then click this button to apply the changes to the list.
- *Remove*: after selecting an existing team, click this button to remove it from the list.

Non-working Periods

The "Non-working Periods" tab is used to define intervals of time on an exception basis where no production will take place, for example holidays and planned maintenance outages:

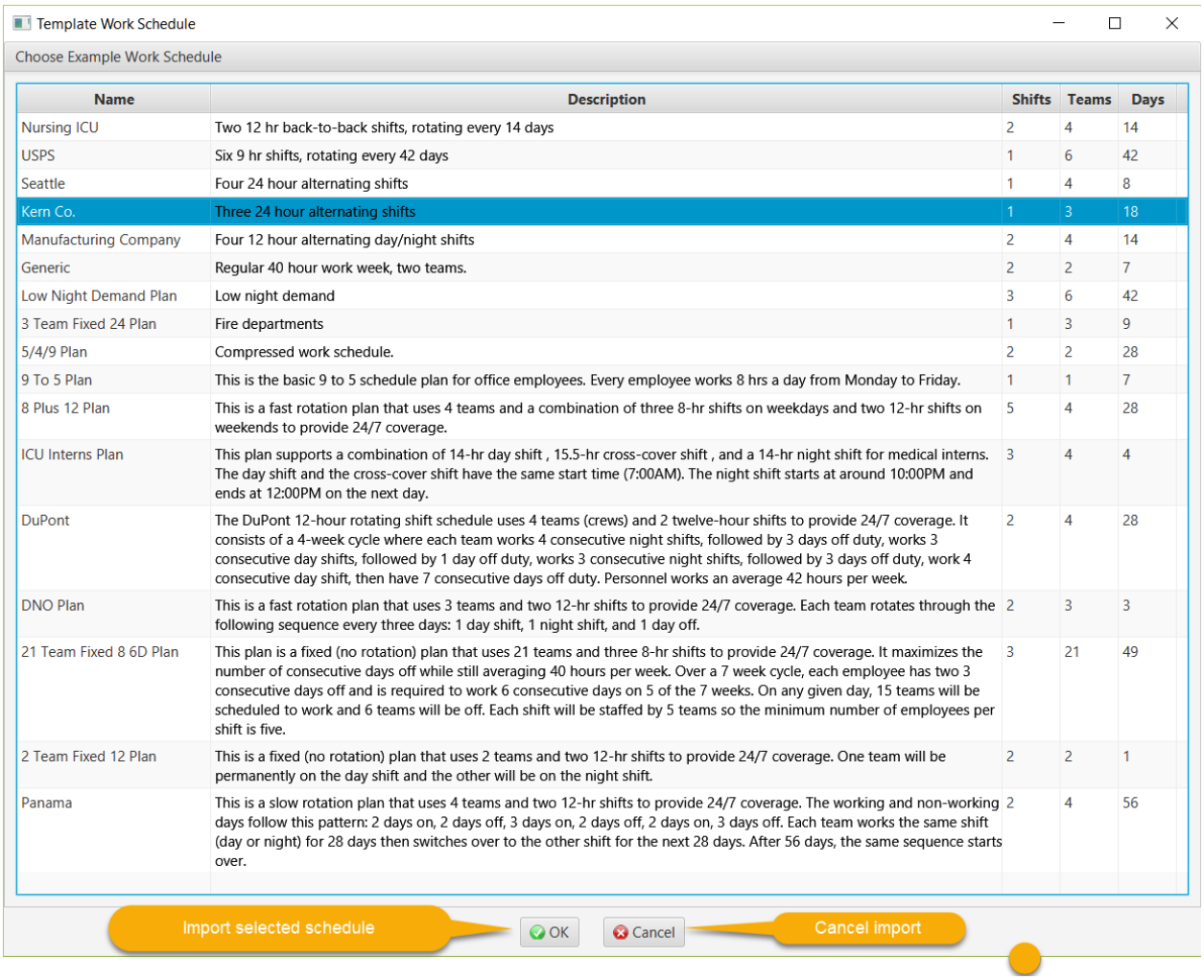


The editor actions for teams are:

- *New*: clear the editor controls in order to define a new non-working period. The Add/Update button text will change to "Add".
- *Add*: After defining the properties of a new non-working period, clicking this button will add it to the list of periods. The loss category is one of two choices (1) Not Scheduled (e.g. holiday), or (2) Unscheduled (special event).
- *Update*: after selecting an existing non-working period, the current values will be moved into the editing controls. Make the necessary edits, then click this button to apply the changes to the list.
- *Remove*: after selecting an existing non-working period, click this button to remove it from the list.

Import Schedule

Click the Import button to launch a dialog to choose a pre-defined work schedule (similar to a desired one) and then save it to the database for further editing:



Name	Description	Shifts	Teams	Days
Nursing ICU	Two 12 hr back-to-back shifts, rotating every 14 days	2	4	14
USPS	Six 9 hr shifts, rotating every 42 days	1	6	42
Seattle	Four 24 hour alternating shifts	1	4	8
Kern Co.	Three 24 hour alternating shifts	1	3	18
Manufacturing Company	Four 12 hour alternating day/night shifts	2	4	14
Generic	Regular 40 hour work week, two teams.	2	2	7
Low Night Demand Plan	Low night demand	3	6	42
3 Team Fixed 24 Plan	Fire departments	1	3	9
5/4/9 Plan	Compressed work schedule.	2	2	28
9 To 5 Plan	This is the basic 9 to 5 schedule plan for office employees. Every employee works 8 hrs a day from Monday to Friday.	1	1	7
8 Plus 12 Plan	This is a fast rotation plan that uses 4 teams and a combination of three 8-hr shifts on weekdays and two 12-hr shifts on weekends to provide 24/7 coverage.	5	4	28
ICU Interns Plan	This plan supports a combination of 14-hr day shift , 15.5-hr cross-cover shift , and a 14-hr night shift for medical interns. The day shift and the cross-cover shift have the same start time (7:00AM). The night shift starts at around 10:00PM and ends at 12:00PM on the next day.	3	4	4
DuPont	The DuPont 12-hour rotating shift schedule uses 4 teams (crews) and 2 twelve-hour shifts to provide 24/7 coverage. It consists of a 4-week cycle where each team works 4 consecutive night shifts, followed by 3 days off duty, works 3 consecutive day shifts, followed by 1 day off duty, works 3 consecutive night shifts, followed by 3 days off duty, work 4 consecutive day shift, then have 7 consecutive days off duty. Personnel works an average 42 hours per week.	2	4	28
DNO Plan	This is a fast rotation plan that uses 3 teams and two 12-hr shifts to provide 24/7 coverage. Each team rotates through the following sequence every three days: 1 day shift, 1 night shift, and 1 day off.	2	3	3
21 Team Fixed 8 6D Plan	This plan is a fixed (no rotation) plan that uses 21 teams and three 8-hr shifts to provide 24/7 coverage. It maximizes the number of consecutive days off while still averaging 40 hours per week. Over a 7 week cycle, each employee has two 3 consecutive days off and is required to work 6 consecutive days on 5 of the 7 weeks. On any given day, 15 teams will be scheduled to work and 6 teams will be off. Each shift will be staffed by 5 teams so the minimum number of employees per shift is five.	3	21	49
2 Team Fixed 12 Plan	This is a fixed (no rotation) plan that uses 2 teams and two 12-hr shifts to provide 24/7 coverage. One team will be permanently on the day shift and the other will be on the night shift.	2	2	1
Panama	This is a slow rotation plan that uses 4 teams and two 12-hr shifts to provide 24/7 coverage. The working and non-working days follow this pattern: 2 days on, 2 days off, 3 days on, 2 days off, 2 days on, 3 days off. Each team works the same shift (day or night) for 28 days then switches over to the other shift for the next 28 days. After 56 days, the same sequence starts over.	2	4	56

SCRIPTING

Description

A JavaScript function must defined for each equipment resolver. This script accepts an input value from an OPC DA, OPC UA, HTTP or messaging data source event and returns a value matching the type of the resolver (e.g. availability, production count, material or job change). The script editor is used to write and test the body of this function.

An availability script outputs the name of a Reason (which is associated with an OEE loss category). A good, reject or startup production script outputs a count in the unit of measure defined for that equipment. A good production amount has the dividend UOM of the design speed, whereas a reject or startup production amount has the defined reject UOM. A material change script outputs the name of a defined material. A job change script outputs the name of a job/order. In order to perform OEE calculations, the material must be defined as the first event before the time period of interest.

The script has three input arguments:

1. OeeContext *context*: an instance of the OeeContext class containing information about the script execution environment (see below for details)
2. Object *value*: the input value
3. EventResolver *resolver*: the event resolver. The resolver's last value can be used in rollover calculations for production counts.

The OeeContext class has these primary public "getter" methods (note that javadocs are available in the "doc" folder):

- *getLogger()*: Get an instance of an org.slf4j.Logger. The logging output(s) is configured in the log4j.properties file
- *getMaterial(Equipment equipment)*: Get the material currently being processed on this equipment. The equipment object is obtained from resolver.getEquipment()
- *getJob(Equipment equipment)*: Get the job currently being run on this equipment. The equipment object is obtained from resolver.getEquipment()
- *getOpcDaClients()*: Get the collection of DaOpcClient objects. A client object can then be used for reading and writing tags, e.g. readSynch()/writeSynch()
- *getOpcUaClients()*: Get the collection of UaOpcClient objects. A client object can then be used for reading and writing nodes, e.g. readSynch()/writeSynch()
- *getMessagingClient()*s: Get the collection of MessagingClient object.s A client object can then be used to send a message to the "Point85" exchange on a RabbitMQ broker, e.g. publish()

The JavaScript below shows examples of calling these methods:

```
// logger
var logger = context.getLogger()


// equipment
var eq = EventResolver()


// log info
logger.info("Material: " + context.getMaterial(eq))
logger.info("Job: " + context.getJob(eq))
logger.info("Source id: " + resolver.getSourceId())
logger.info("Last value: " + resolver.getLastValue())
logger.info("Last timestamp: " + resolver.getLastTimestamp())
logger.info(context.toString())
```

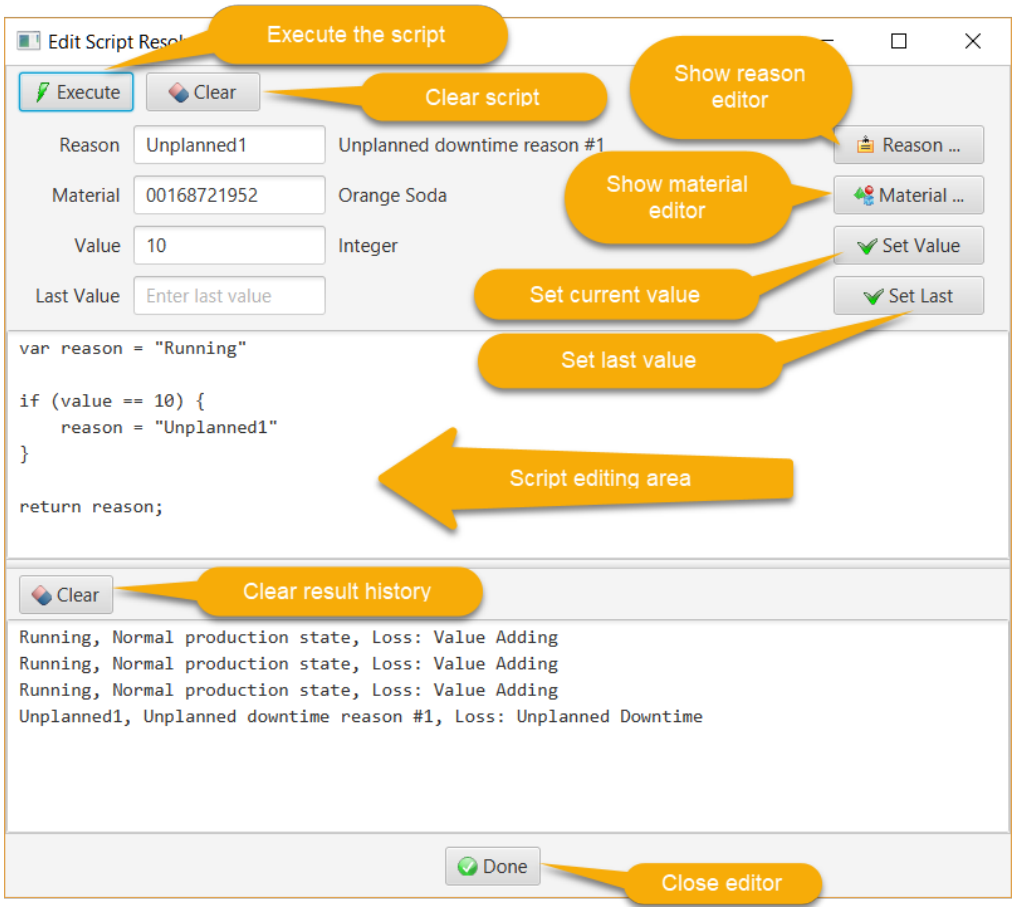
```
// send RMQ message

var CollectorNotificationMessage =
Java.type("org.point85.domain.messaging.CollectorNotificationMessage")

var routingKey = Java.type("org.point85.domain.messaging.RoutingKey").NOTIFICATION_MESSAGE

var severity = Java.type("org.point85.domain.messaging.NotificationSeverity").INFO

var msg = new CollectorNotificationMessage("localhost", "192.168.0.8")

msg.setText("This is a notification")

msg.setSeverity(severity)

context.getMessagingClient().publish(msg, routingKey, 30)


// OPC DA read integer value

var variant = context.getOpcDaClient().readSynch("Saw-toothed Waves.Int2")

logger.info("Value: " +  variant.getValueAsNumber())


// OPC DA write integer value

var OpcDaVariant = Java.type("org.point85.domain.opc.da.OpcDaVariant")

var variant = new OpcDaVariant(100)

context.getOpcDaClient().writeSynch("Data Type Examples.16 Bit Device.K Registers.Short1",
variant)


// OPC UA read current server time

var NodeId = Java.type("org.eclipse.milo.opcua.stack.core.types.builtin.NodeId")

var nodeId = new NodeId(0, 2258)

var dataValue = context.getOpcUaClient().readSynch(nodeId)

logger.info(dataValue.getValue().getValue())


// OPC UA write integer value

var NodeId = Java.type("org.eclipse.milo.opcua.stack.core.types.builtin.NodeId")

var Variant = Java.type("org.eclipse.milo.opcua.stack.core.types.builtin.Variant")

var nodeId = new NodeId(3, "Int32DataItem")

var value = new Variant(100)

code = context.getOpcUaClient().writeSynch(nodeId, value)

if (code.isBad()) {

       logger.error("Write failed, code = " + code.getValue())

}


// database query
```

```
var service = Java.type("org.point85.domain.persistence.PersistenceService").instance()
var sql = "Select top 10 EVENT_TYPE, JOB from OEE_EVENT order by START_TIME desc"
var rows = service.getEntityManager().createNativeQuery(sql).getResultList()
for (i = 0; i < rows.size(); i++) {
        var row = rows.get(i)
        logger.info("Type: " + row[0] + ", Job: " + row[1])
}
```

Editor

The script editor dialog is launched either by (1) selecting an equipment resolver in the list in the "Data Collection" tab of the equipment entity and then clicking the editor button or (2) clicking a button on the toolbar. If the editor is launched from the toolbar, functionality will be limited since the resolver input argument to the script will be null.

The editor looks like:



The editor actions are:

- *Execute*: execute the script. The returned object's toString() method will be called and the result displayed in the execution history in the bottom pane.

- *Clear*: clear the script editor or history
- *Reason*...: Display the reason editor to create or update a reason and choose an existing reason. The name is displayed in the text field where it is available for cutting and pasting into the script.
- *Material*...: Display the material editor to create or update a material and choose an existing material. The name is displayed in the text field where it is available for cutting and pasting into the script.
- *Set Value*: Set the value in the text field as the input value to the script before it is executed.
- *Set Last*: Set the value in the text field as the previous input value to the script before it is executed.

Depending upon the type of resolver, the script area initially will be populated by a default script. Availability, material change and job change scripts pass the input value through to the output:

```
return value;
```

whereas a production count script provides a variable called "ROLLOVER" as a place-holder to take into account the case where a counter can output a lower value than a previous value:

```
var ROLLOVER = 0;
var lastValue = resolver.getLastValue();
var delta = value - lastValue;
if (value < lastValue) {
    delta += ROLLOVER;
}
return delta;
```

DATA SOURCE EDITORS

The data source editors define where a non-manual input value to a resolver script can come from. The supported sources are:

- OPC DA: classic OLE for Process Control (OPC) Data Acquisition
- OPC UA: OLE for Process Control Unified Architecture (UA)
- HTTP: invocation of an HTTP POST request with the data in the request body
- Messaging: an event message received via a RabbitMQ message broker with the data as the message payload

OPC UA Data Source

Browser

The OPC UA data source browser dialog is launched from the toolbar or from the Data Collection tab after an equipment object with an OPC UA source has been selected in the physical model. It is used to browse to the node providing the input value. This dialog looks like:

In this example, the browser is anonymously connected to the Prosys demo server running on a host at 192.168.0.13 IP address on port 53530 with a server name/path of OPCUA/SimulationServer.

If a secure connection is desired for a server, under the Security Settings tab, the Security Policy can be chosen from None, Basic128Rsa15, Basic256, Basic256Sha256, Aes128_Sha256_RsaOaep or Aes256_Sha256_RsaPss. The Message Security Mode can be chosen from None, Sign or Sign & Encrypt. Under the Authentication Settings tab, the Java keystore file name can be specified in the Keystore text field and its password in the Password text field. The keystore file must be placed in the config/security folder. The user name and user password text fields can be used to specify the user name and password for user authentication. The button to the right of the keystore file name clears out these security settings.

The actions for an OPC UA data source are:

- *New*: clear the editing controls to define a new data source

- *Save*: save the data source to the database
- *Delete*: delete the data source from the database

The actions for establishing a connection are:

- *Connect*: connect to the data source
- *Disconnect*: disconnect from the data source
- *Cancel*: cancel an unsuccessful connection attempt

After a connection is established, the server namespace can be browsed. Selection of a node will display the current value and information about it below the tree view. In this example, the node namespace is 5 with a string id of "Triangle1".

Clicking the Done button will assign the selected node as the script resolver's input source.

Trending

Suppose that an OPC UA event resolver executes an availability script for a Unified Automation OPC UA demo server. An availability script must output a reason. For this node (triangle trend for a double value), the publishing interval is every 5 seconds. The script outputs a reason based on the input value:

```
var reason = 'Running';
if (value < 0.0)
{
    reason = 'Unplanned1';
}
return reason;
```

By clicking the Watch button for this resolver, the execution of the script can be observed:

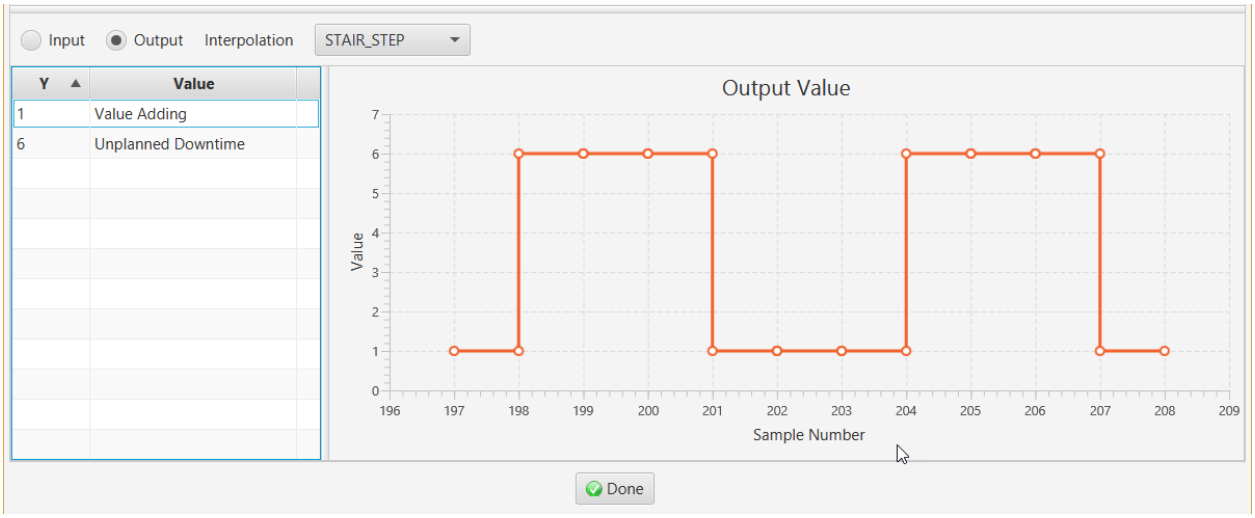
The actions are:

- *Connect*: connect to the OPC UA data source. After a successful connection, the server id and node ids are displayed along with the server status (Running in this case).
- *Disconnect*: disconnect from the data source
- *Cancel*: abort an unsuccessful connection attempt
- *Stop/Start*: The trending can be paused by clicking this button. The text will change to Start. The trend can be restarted by clicking it again.
- *Reset*: Restart trending after changing either the update period or number of points to display.

The table shows the item id, input value, timestamp and output value. If the output is an availability reason, the time loss category is displayed (Unplanned Downtime in this case).

The current update period and number of data points to display on the X axis is displayed. These values can be changed when the trend is stopped and take effect after it is reset and restarted.

If Output is selected for the trend, the chart looks like:

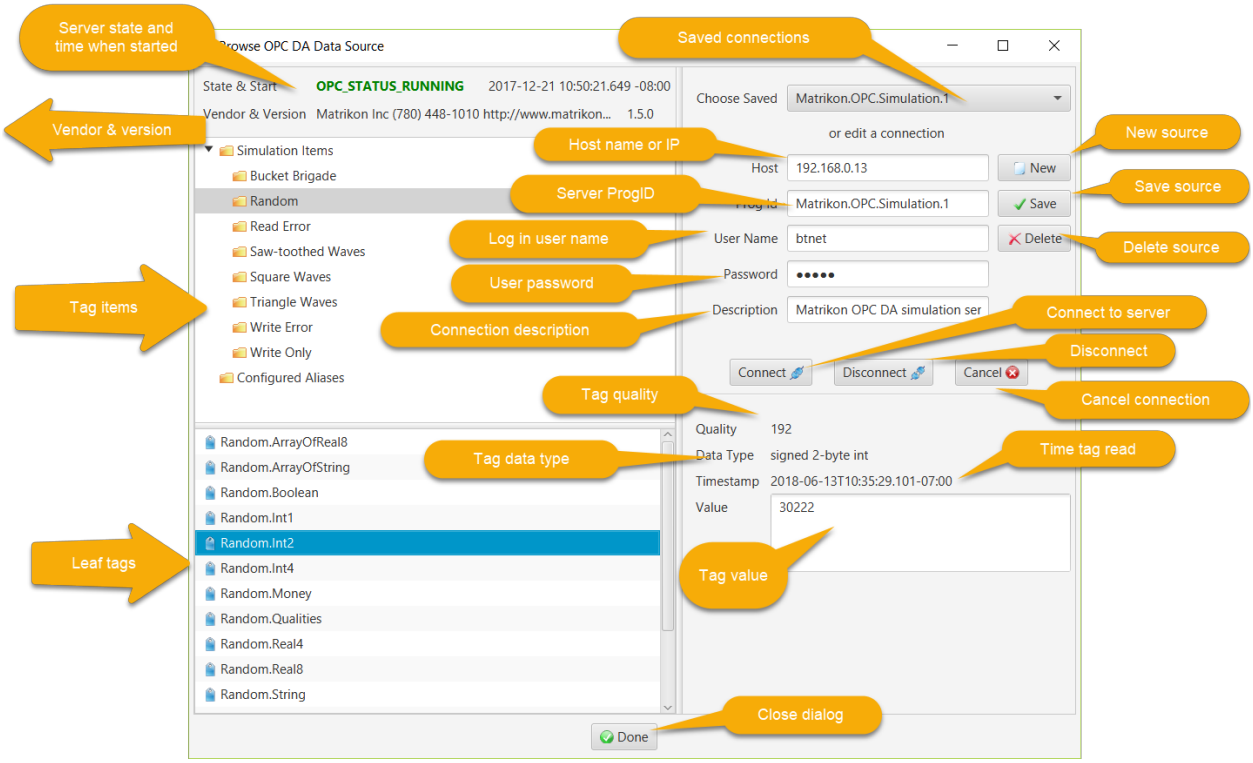


Since the output is a reason, the values are discrete and thus a stair step interpolation is desired (the screen capture above displays a linear interpolation). The table on the left shows an integral value for the Y axis and the corresponding discrete value. Linear interpolation applies to continuous values such as integer or floating point data.

OPC DA Data Source

Browser

The OPC DA data source browser dialog is launched from the toolbar or from the Data Collection tab after an equipment object with an OPC DA data source has been selected in the physical model. It is used to browse to the tag providing the input value. This dialog looks like:



In this example, the browser is connected to the Matrikon OPC simulation server on host 192.168.0.13 with a ProgID of Matrikon.OPC.Simulation.1 and the “btnet” user and password (note that the user name can include a Windows domain name).

The actions for an OPC DA data source browser are:

- *New*: clear the editing controls to define a new data source
- *Save*: save the data source to the database
- *Delete*: delete the data source from the database

The actions for establishing a connection are:

- *Connect*: connect to the data source
- *Disconnect*: disconnect from the data source
- *Cancel*: cancel an unsuccessful connection attempt

After a connection is established, the server tags can be browsed. Selection of a parent item of a leaf tag will display the children below the tree view. Selecting a leaf tag will display the tag’s current value and information about it to the right of the tree view.

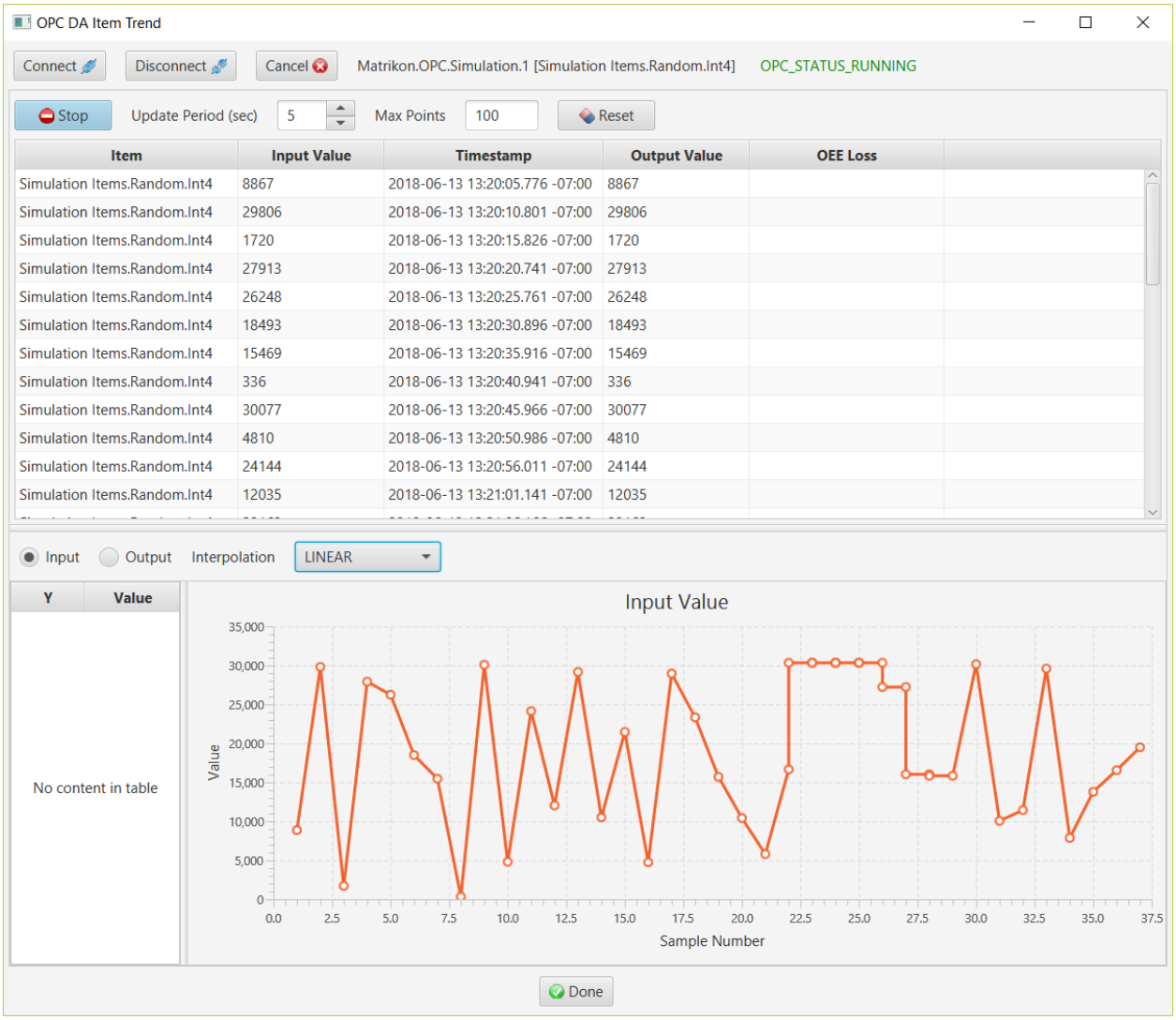
Clicking the Done button will assign the selected tag as the OPC DA script resolver’s input source.

Trending

By clicking the Watch button for an OPC DA resolver, the execution of the script can be observed. A trend chart for an OPC DA source for a 4-byte integer good production count with a pass-through resolver script of:

```
return value;
```

looks like:

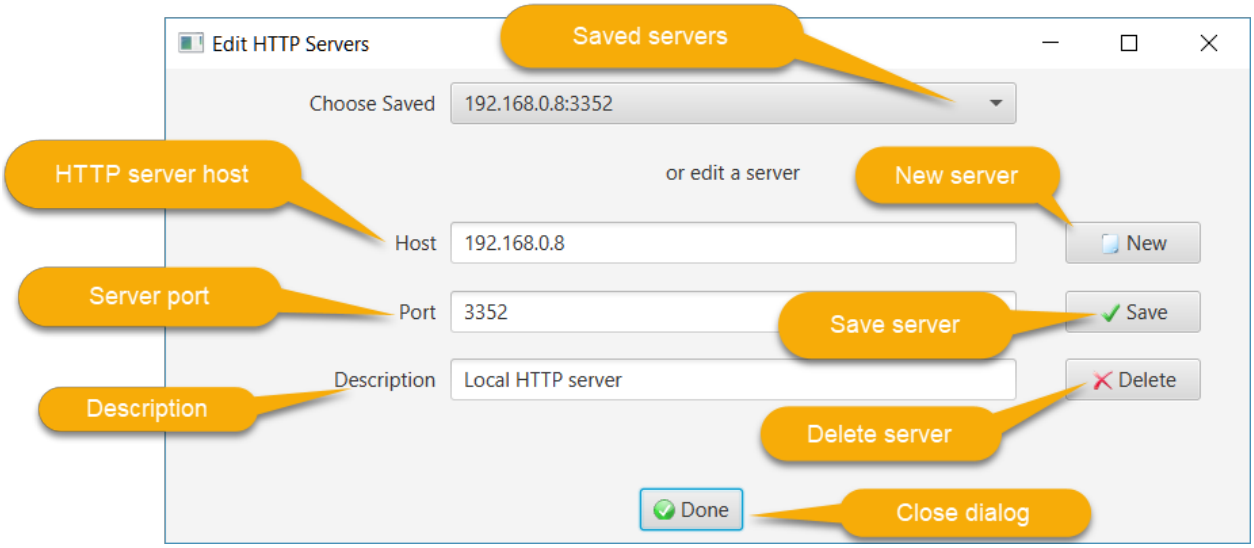


HTTP Data Source

Definition

The HTTP data source definition dialog is launched from the toolbar or from the Data Collection tab after an equipment object with an HTTP source has been selected in the physical model. It is used to define the host and port for the NanoHTTPD embedded HTTP server in the data collector.

This dialog looks like:



The actions for an HTTP data source are:

- *New*: clear the editing controls to define a new data source
- *Save*: save the data source to the database
- *Delete*: delete the data source from the database

Clicking the Done button will assign the HTTP server as the script resolver's input source.

When the data collector is started on the specified host (192.168.0.8 in this example), it will listen to the specified port (3352) ready to receive POST requests.

The POST request has the EquipmentEventRequestDto JSON serialized DTO (Data Transfer Object) as a body (sourceId, value and timestamp fields). This request is for an availability, production of material, setup or job change event.

The HTTP server responds with the corresponding EquipmentEventResponseDto JSON body (status and errorText fields).

Java Client Example

An HTTP Java client can post an equipment event request. For example to loop-back test in the HTTP trend dialog executes this code:

```
@FXML
private void onLoopbackTest() {
        HttpURLConnection conn = null;
        try {
                // get the HTTP data source
                EventResolver eventResolver = trendChartController.getEventResolver();
                HttpSource dataSource = (HttpSource) eventResolver.getDataSource();

                // build the URL for an equipment event
                URL url = new URL(
                        "http://" + dataSource.getHost() + ":" + dataSource.getPort() + '/' +
OeeHttpServer.EVENT_EP);

                // create a connection for a JSON POST request
```

```
                conn = (HttpURLConnection) url.openConnection();
                conn.setDoOutput(true);
                conn.setRequestMethod("POST");
                conn.setRequestProperty("Content-Type", "application/json");

                // the value to send (must match the configured resolver)
                String value = tfLoopbackValue.getText();

                // timestamp when sent
                String timestamp = DomainUtils.offsetDateTimeToString(OffsetDateTime.now());

                // create the data transfer event object
                EquipmentEventRequestDto dto = new EquipmentEventRequestDto(eventResolver.getSourceId(),
value, timestamp);

                // serialize the body
                Gson gson = new Gson();
                String payload = gson.toJson(dto);

                // make the request
                OutputStream os = conn.getOutputStream();
                os.write(payload.getBytes());
                os.flush();

                if (logger.isInfoEnabled()) {
                        logger.info("Posted equipment event request to URL " + url + " with value " + value);
                }

                // check the response code
                int codeGroup = conn.getResponseCode() / 100;

                if (codeGroup != 2) {
                        String msg = "Post failed, error code : " + conn.getResponseCode() + "\nEquipment
event response ...";
                        BufferedReader br = new BufferedReader(new
InputStreamReader((conn.getInputStream())));
                        String output;

                        while ((output = br.readLine()) != null) {
                                msg += "\n" + output;
                        }
                        throw new Exception(msg);
                }
        } catch (Exception e) {
                AppUtils.showErrorDialog(e);
        } finally {
                conn.disconnect();
        }
}
```

Database Trigger Example

For another example, a database table insertion trigger can be used to asynchronously post equipment event messages to an HTTP collector. For example, SQL Server supports creating a stored procedure in C#. This procedure can then be executed in a trigger. The C# codes makes the HTTP request and receives the response. For simplicity, the values inserted into an EQUIPMENT_EVENT table row will be input to the stored procedure and then posted to the HTTP collector at the specified URL.

The EQUIPMENT_EVENT data table is created as:

```
        CREATE TABLE [dbo].[EQUIPMENT_EVENT] (

            [Id]         INT                NOT NULL,

            [SOURCE_ID]  NVARCHAR (64)      NOT NULL,

            [VALUE]      NVARCHAR (32)      NOT NULL,

            [EVENT_TIME] DATETIMEOFFSET (3) NOT NULL

        );
```

Suppose that a pass-through availability script resolver with source id = "e1.avail" and data value = "r1" has been created. "r1" is a reason with a loss category. When the following row is inserted into the EQUIPMENT_EVENT table, we want to call the stored procedure to make the HTTP request:

```
insert into EQUIPMENT_EVENT (Id, SOURCE_ID, VALUE, EVENT_TIME) values (1, 'e1.avail', 'r1', SYSDATETIMEOFFSET())
```

The insertion database trigger for the table is created as:

```
CREATE TRIGGER [ON_EVENT]
	ON [dbo].[EQUIPMENT_EVENT]
	FOR INSERT
	AS
	BEGIN
		SET NOCOUNT ON
		-- event endpoint
		declare @url nvarchar(128)
		set @url = 'http://machine_ip:8184/event'
		declare @response nvarchar(1024)
		declare @sourceId nvarchar(64)
		declare @value nvarchar(64)
		declare @timestamp datetimeoffset(3)
		declare @event_time nvarchar(64)
		select @sourceId = i.SOURCE_ID, @value = i.VALUE, @timestamp = i.EVENT_TIME from inserted i
		select @event_time = convert(nvarchar(64), @timestamp, 126)
		exec PostEquipmentEvent @url, @sourceId, @value, @event_time, @response output
	END
```

Here the HTTP data collector is running at "machine_ip" address on port 8184. The data to be sent to the collector is obtained from the "inserted" row and then passed into the PostEquipmentEvent stored procedure. The collector's JSON response is returned in the @response output parameter.

The PostEquipmentEvent stored procedure is written in C# as:

```
public partial class StoredProcedures
{
    [Microsoft.SqlServer.Server.SqlProcedure]
    public static void PostEquipmentEvent(string url, string sourceId, string value, string
timestamp, out string result)
    {
            // POST equipment event
            // json content
```

```
                string content = "{\"sourceId\":\"" + sourceId + "\",\"value\":\"" + value +
"\",\"timestamp\":\"" + timestamp + "\"}";

                // create Http request

                HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);

                byte[] data = Encoding.ASCII.GetBytes(content);

                request.Method = "POST";

                request.ContentType = "application/json";

                request.ContentLength = data.Length;

                // make the request

                Stream postStream = request.GetRequestStream();

                postStream.Write(data, 0, data.Length);

                // wait for the response

                HttpWebResponse response = (HttpWebResponse)request.GetResponse();

                result = new StreamReader(response.GetResponseStream()).ReadToEnd();

                response.Close();

                postStream.Close();

        }

}
```

iOS/Android Example

The HTTP URL can be called from an IOS or Android mobile application. In this case, the user interface is built using the native IDE (XCode and Swift for iOS, Android Studio and Java for Android). An HTTP client API is then called to make a request and receive a response.

For example, a Swift function for a POST request is:

```
    // send an HTTP POST request with body data

    private func sendPostRequest(_ url: String, body: String) -> NSError? {

        if let error = validateRequest() {

            return error

        }

        let nsUrl = URL(string : url)!

        var request = URLRequest(url: nsUrl)

        request.httpMethod = "POST"

        let bodyData : Data = body.data(using: String.Encoding.utf8)!

        request.httpBody = bodyData

        dataTask = dataSession.dataTask(with: request, completionHandler: {

            data, response, error in
```

```
            // flag that task is done
            self.dataTask = nil
            // call back handler
            self.handler!.handleResponse(nsUrl, data: data, error: error)
        })
        dataTask?.resume()
        return nil
    }
```

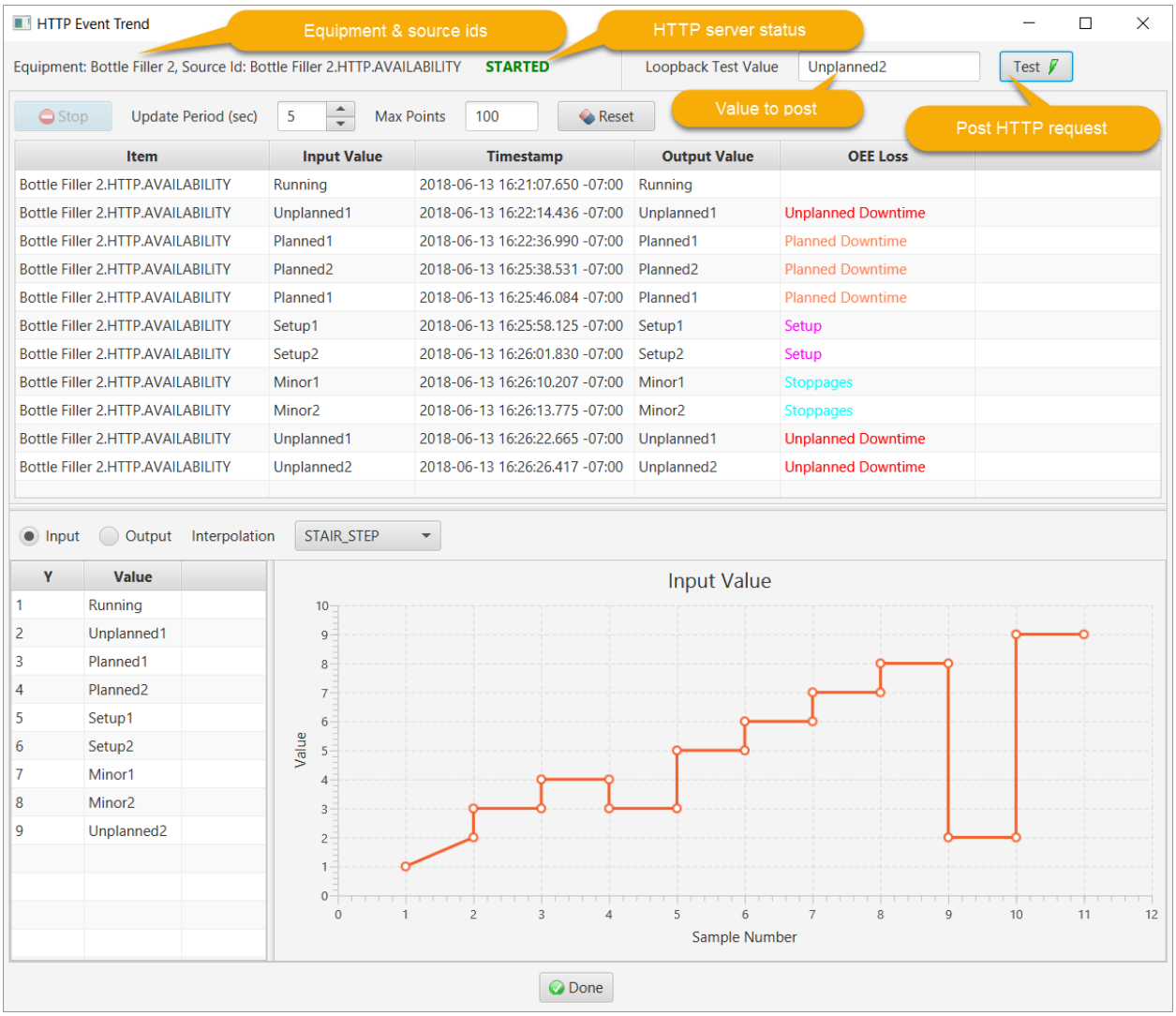
The caller of this function provides the URL with the endpoint (e.g. “event”) and the JSON serialized body.

Trending

By clicking the Watch button for an HTTP resolver, the execution of the script can be observed. An HTTP source for equipment availability with a pass-through resolver script of:

```
return value;
```

looks like:



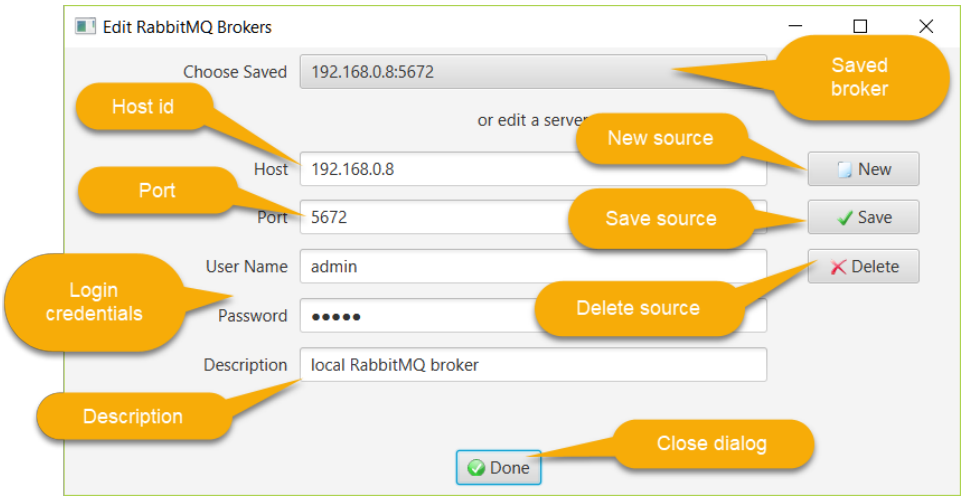
In this example, reasons have been entered into the loop-back test field and the Test button clicked to send a POST request to the HTTP server embedded in the controller for this dialog.

Messaging Data Source

Definition

The messaging data source definition dialog is launched from the toolbar or from the Data Collection tab for a messaging resolver after an equipment object has been selected in the physical model. It is used to define the RabbitMQ broker host, port and login credentials.

This dialog looks like:



For this example, the RabbitMQ broker is running on host 192.168.0.8 on the default port of 5672. The client will login as the "admin" user.

The actions for a messaging data source are:

- *New*: clear the editing controls to define a new data source
- *Save*: save the data source to the database
- *Delete*: delete the data source from the database

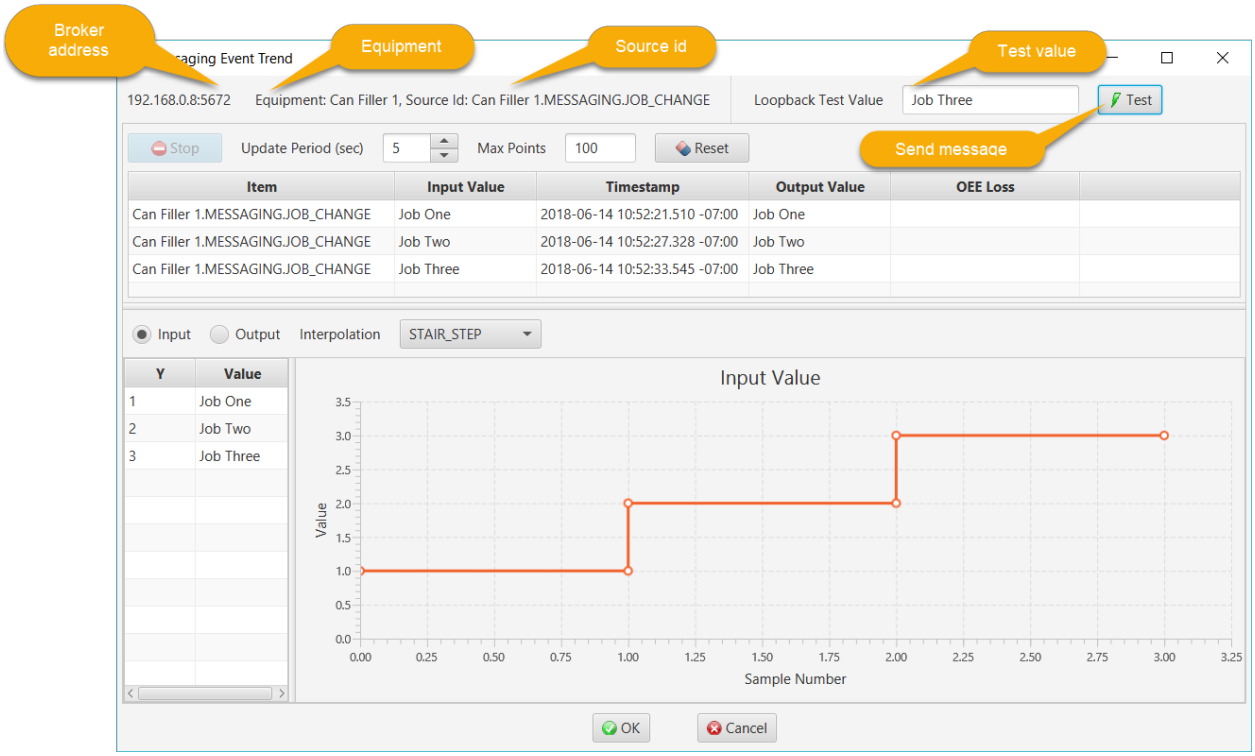
Clicking the Done button will assign the broker as the script resolver's input source.

Trending

By clicking the Watch button for an messaging resolver, the execution of the script can be observed. A trend for a messaging source for a job change with a pass-through resolver script of:

```
return value;
```

looks like:



In this example, job identifiers have been entered into the loop-back test field and the Test button clicked to send a JSON serialized EquipmentEventMessage to the specified RabbitMQ broker. The messaging trend controller is listening for these messages from the Point85 exchange and routed to its queue.

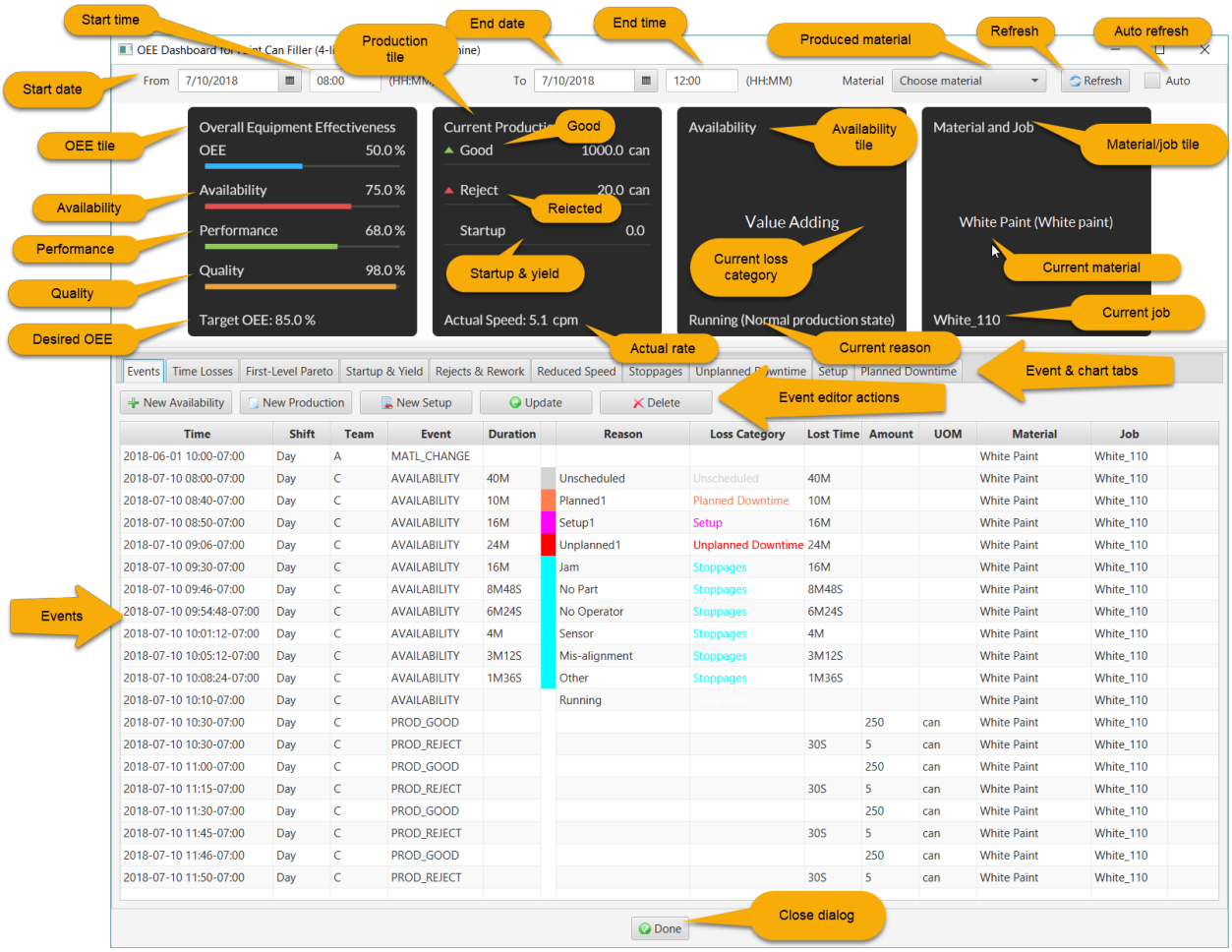
The Java code is:

```
@FXML
private void onLoopbackTest() {
        try {
                if (pubSub == null) {
                        throw new Exception("The trend is not connected to an RMQ broker.");
                }
                EventResolver eventResolver = trendChartController.getEventResolver();
                String sourceId = eventResolver.getSourceId();
                String value = tfLoopbackValue.getText();
                EquipmentEventMessage msg = new EquipmentEventMessage();
                msg.setSourceId(sourceId);
                msg.setValue(value);
                pubSub.publish(msg, RoutingKey.EQUIPMENT_SOURCE_EVENT, 30);
        } catch (Exception e) {
                AppUtils.showErrorDialog(e);
        }
```

}

DASHBOARD

After equipment is selected, the dashboard can be displayed. This form consists of four tiles at the top that display OEE, availability, production counts, material and job. The bottom portion has tabs for event history, time losses by category and various Pareto charts:



This dashboard is for a paint can filling machine as discussed in [Kennedy]. A time range is first selected from a starting date and time of day to and ending date and time of day. In the above example, the time period is 4 hours during a single day. A specific material that has setups can be selected in the combobox, or all materials with setups during the specified time period. Clicking the refresh button will query the event records from the database and compute the OEE with its three components. In this case, the OEE is 50% where availability = 75%, performance = 68% and quality = 98%. The paint can fillter machine has a desired OEE of 85%. If the "Auto" checkbox is checked, the form will refresh periodically.

The current production tile displays the cumulative good (1000 cans), reject (20 cans) and startup counts (0 cans) from the 8 event records.

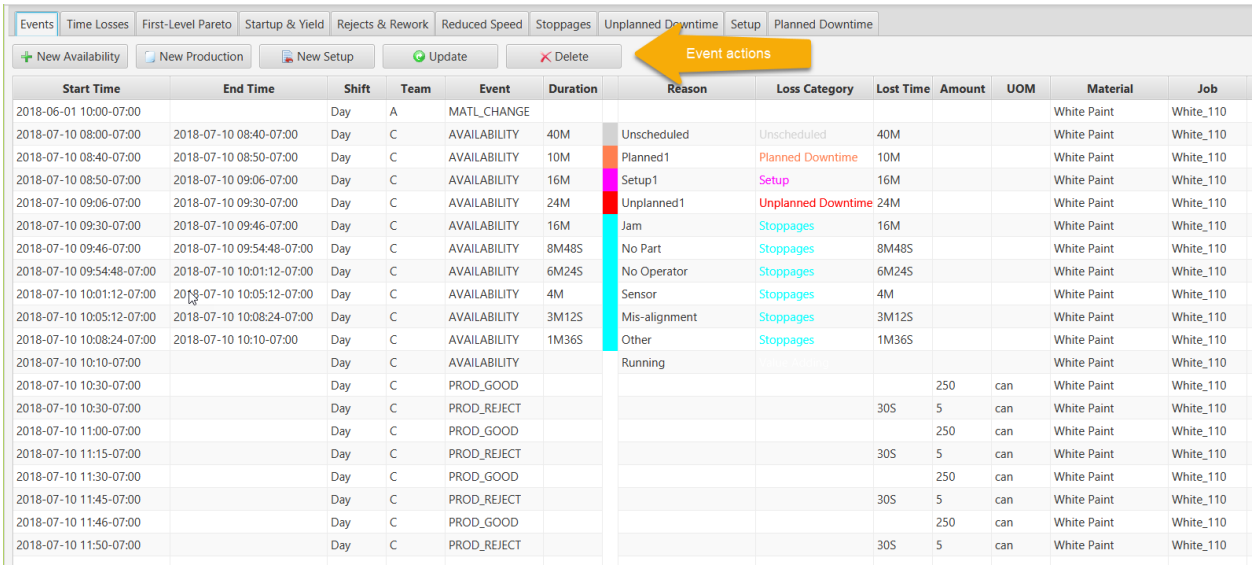
The availability tile displays the last availability event (a Value Adding loss category) with a reason of "Running" and description. Note that in this example, the availability events were entered in temporal sequence for a 4 hour period starting at 8 am and ending at 12 noon. It is also possible to enter these events in summary form (where the event duration includes multiple events for the same reason).

The material and job tile displays the current material being run (White Paint) and job name (White_110).

Below the tiles the following tabs have more detailed information.

Events

This tab displays the event history in a table:



Start Time	End Time	Shift	Team	Event	Duration	Reason	Loss Category	Lost Time	Amount	UOM	Material	Job
2018-06-01 10:00-07:00		Day	A	MATL_CHANGE							White Paint	White_110
2018-07-10 08:00-07:00	2018-07-10 08:40-07:00	Day	C	AVAILABILITY	40M	Unscheduled	Unscheduled	40M			White Paint	White_110
2018-07-10 08:40-07:00	2018-07-10 08:50-07:00	Day	C	AVAILABILITY	10M	Planned1	Planned Downtime	10M			White Paint	White_110
2018-07-10 08:50-07:00	2018-07-10 09:06-07:00	Day	C	AVAILABILITY	16M	Setup1	Setup	16M			White Paint	White_110
2018-07-10 09:06-07:00	2018-07-10 09:30-07:00	Day	C	AVAILABILITY	24M	Unplanned1	Unplanned Downtime	24M			White Paint	White_110
2018-07-10 09:30-07:00	2018-07-10 09:46-07:00	Day	C	AVAILABILITY	16M	Jam	Stoppages	16M			White Paint	White_110
2018-07-10 09:46-07:00	2018-07-10 09:54:48-07:00	Day	C	AVAILABILITY	8M48S	No Part	Stoppages	8M48S			White Paint	White_110
2018-07-10 09:54:48-07:00	2018-07-10 10:01:12-07:00	Day	C	AVAILABILITY	6M24S	No Operator	Stoppages	6M24S			White Paint	White_110
2018-07-10 10:01:12-07:00	2018-07-10 10:05:12-07:00	Day	C	AVAILABILITY	4M	Sensor	Stoppages	4M			White Paint	White_110
2018-07-10 10:05:12-07:00	2018-07-10 10:08:24-07:00	Day	C	AVAILABILITY	3M12S	Mis-alignment	Stoppages	3M12S			White Paint	White_110
2018-07-10 10:08:24-07:00	2018-07-10 10:10-07:00	Day	C	AVAILABILITY	1M36S	Other	Stoppages	1M36S			White Paint	White_110
2018-07-10 10:10-07:00		Day	C	AVAILABILITY		Running					White Paint	White_110
2018-07-10 10:30-07:00		Day	C	PROD_GOOD					250	can	White Paint	White_110
2018-07-10 10:30-07:00		Day	C	PROD_REJECT				30S	5	can	White Paint	White_110
2018-07-10 11:00-07:00		Day	C	PROD_GOOD					250	can	White Paint	White_110
2018-07-10 11:15-07:00		Day	C	PROD_REJECT				30S	5	can	White Paint	White_110
2018-07-10 11:30-07:00		Day	C	PROD_GOOD					250	can	White Paint	White_110
2018-07-10 11:45-07:00		Day	C	PROD_REJECT				30S	5	can	White Paint	White_110
2018-07-10 11:46-07:00		Day	C	PROD_GOOD					250	can	White Paint	White_110
2018-07-10 11:50-07:00		Day	C	PROD_REJECT				30S	5	can	White Paint	White_110

The starting and ending date and times of day are displayed. Current events do not have an ending time (e.g. the setup that changed to the current material and job).

If a work schedule is defined for this equipment, the shift and team working that shift will be shown. The duration of an event is the availability time period or the total time period for production events. Therefore, for availability events, the lost time is equal to the duration. But, for reject or startup production, the counts are converted into the equivalent time period as per the OEE time loss model.

The availability reason is shown and its color-coded loss category. Production events have the amount and unit of measure. The last table columns show the material and job when the event ocurred.

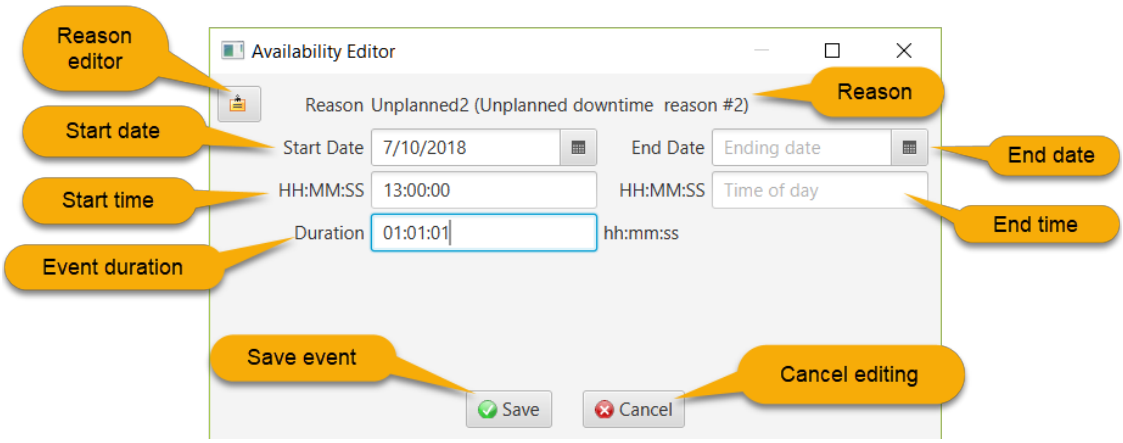
The following actions are available:

- *New Availability*: create an availability record. The dialog discussed below will be displayed to enter the event information.
- *New Production*: create a production record. The dialog discussed below will be displayed to enter the event information.

- *New Setup*: create a set up record. The dialog discussed below will be displayed to enter the event information.
- *Update*: edit an existing availability, production or setup record. The dialog discussed below will be displayed to enter the event information.
- *Delete*: delete an existing set up record.

Availability Editor

To create an availability event or to edit an existing event, the following dialog is used:



If an existing event is being edited, the fields will be populated with current data.

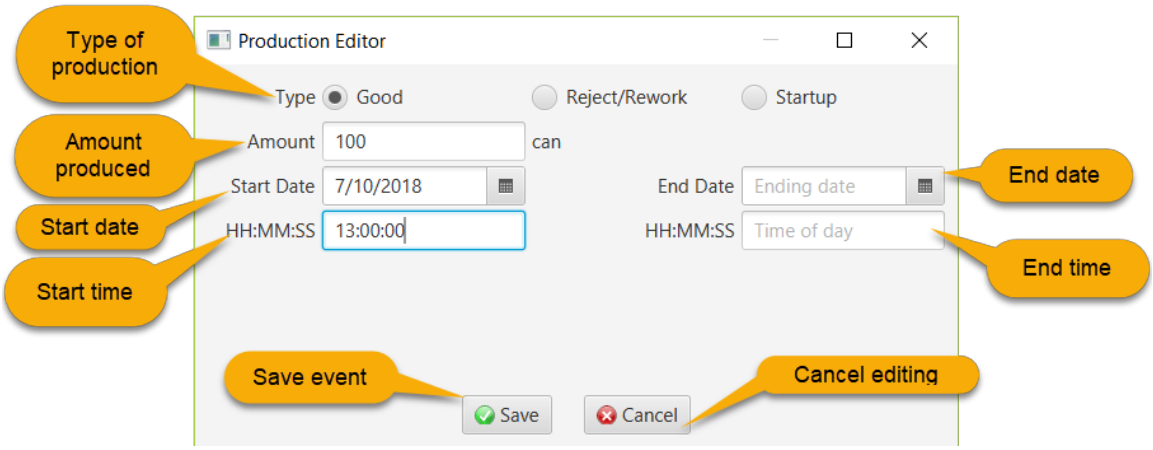
A reason must be chosen by clicking on the reason editor button and selecting an existing reason or creating a new reason. A starting date and time of day is required as is the duration of the event.

An ending time is not required if the events are being entered in chronological order as they happened. On the other hand, if the event is being summarized over a period of time, the ending date and time of day is required.

Clicking the Save button inserts a record into the database. Cancel will exit the editor without making any changes.

Production Editor

To create a production event or to edit an existing event, the following dialog is used:



If an existing event is being edited, the fields will be populated with current data.

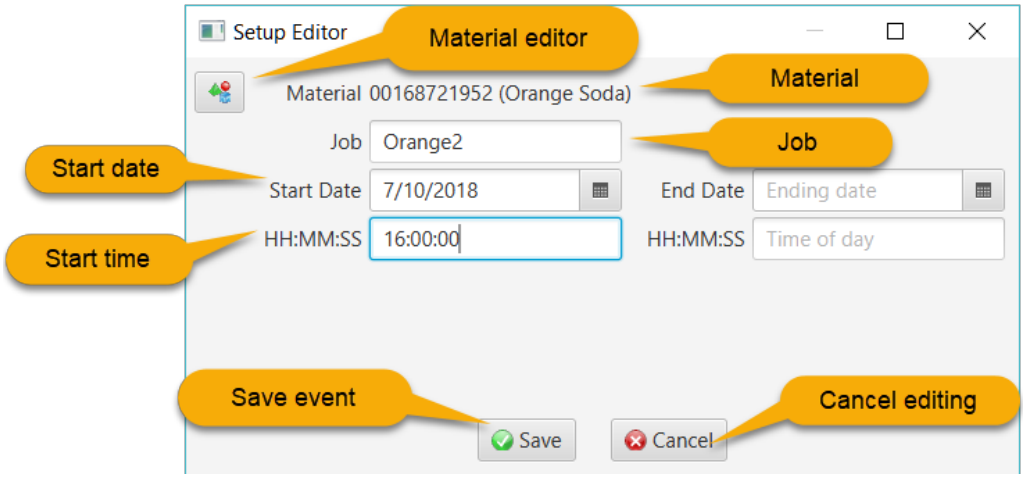
A type of production must be selected in the radio buttons and the amount produced. The unit of measure is obtained from the UOM configured for the equipment.

An ending time is not required if the events are being entered in chronological order as they happened. On the other hand, if the event is being summarized over a period of time, the ending date and time of day is required.

Clicking the Save button inserts a record into the database. Cancel will exit the editor without making any changes.

Setup Editor

To create a setup event or to edit an existing event, the following dialog is used:



If an existing event is being edited, the fields will be populated with current data.

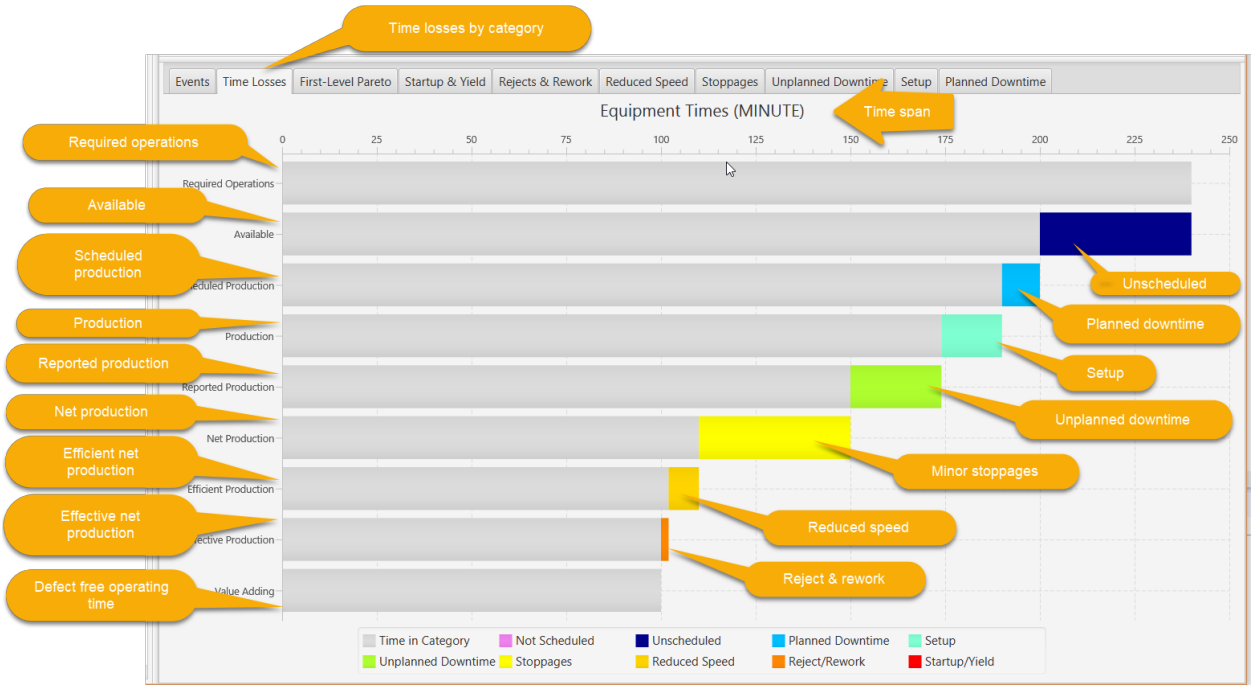
The material must be chosen by clicking on the material editor button and selecting an existing material or creating a new material. The name of a job/order is optional. A starting date and time of day is required.

An ending time is not required if the events are being entered in chronological order as they happened. On the other hand, if the event is being summarized over a period of time, the ending date and time of day is required. Note that there must be at least one open setup event since OEE data is dependent upon knowing what material is being produced.

Clicking the Save button inserts a record into the database. Cancel will exit the editor without making any changes.

Time Losses

The time losses tab displays a bar chart of of the losses encountered before arriving at the net defect free or value adding time (see [Kennedy]).



The bars represent:

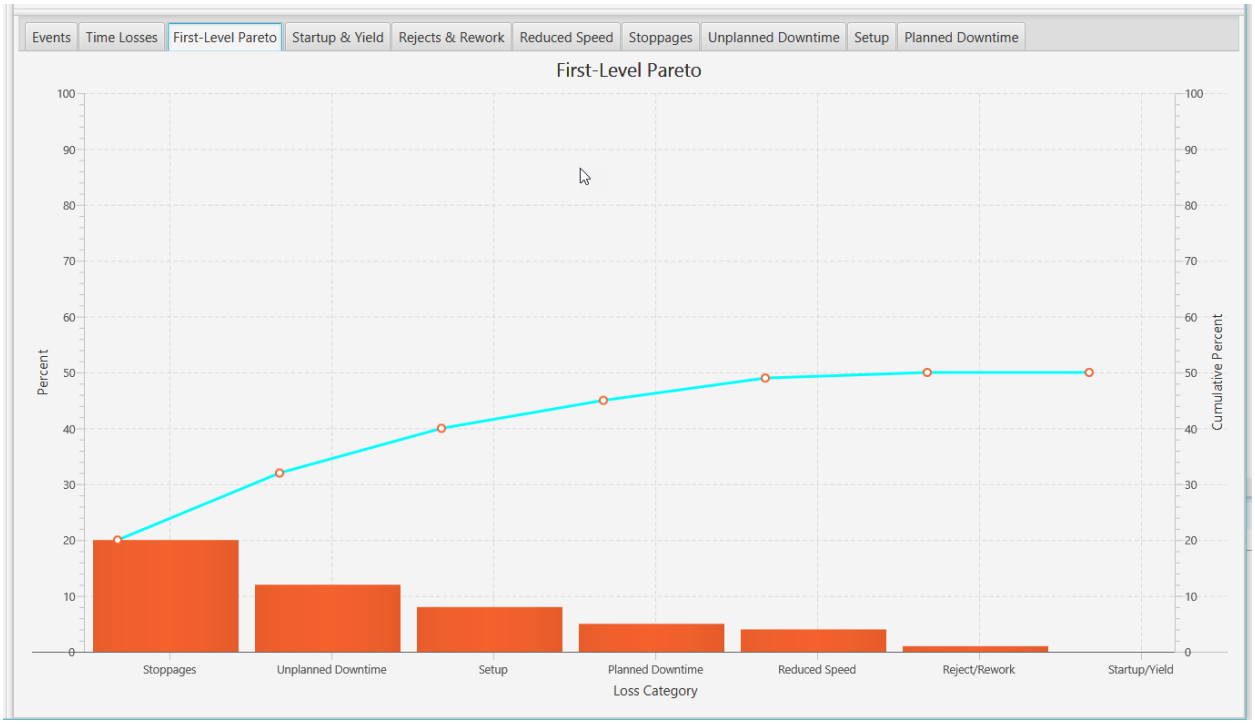
- *Required Operations*: the time period of interest (4 hours or 240 minutes in this example) minus the Not Scheduled time (also 4 hours since there is no Not Scheduled time)
- *Available*: the required production time that subtracts the Unscheduled time (40 minutes)
- *Scheduled Production*: available time less Planned Downtime (10 minutes)
- *Production*: scheduled production time less the Setup time (16 minutes)
- *Reported Production*: production time less the Unplanned Downtime time (24 minutes)
- *Net Production*: reported production time less the Minor Stoppages time (40 minutes)
- *Efficient Net Production*: net production time less the Reduced Speed time (8 minutes)
- *Effective Net Production*: effecient net production time less the Reject/Rework time (2 minutes)

- *Value Adding, Defect Free or Ideal Speed*: effective net production time less the Startup & Yield time (0 minutes)

First-Level Pareto

The first-level pareto chart displays the percentage of Available Time consumed by each of the 7 OEE losses:

1. Planned Downtime
2. Setup
3. Unplanned Downtime
4. Minor Stoppages
5. Reduced Speed
6. Rejects and Rework
7. Startup and Yield



COLLECTOR APPLICATION

MONITOR APPLICATION

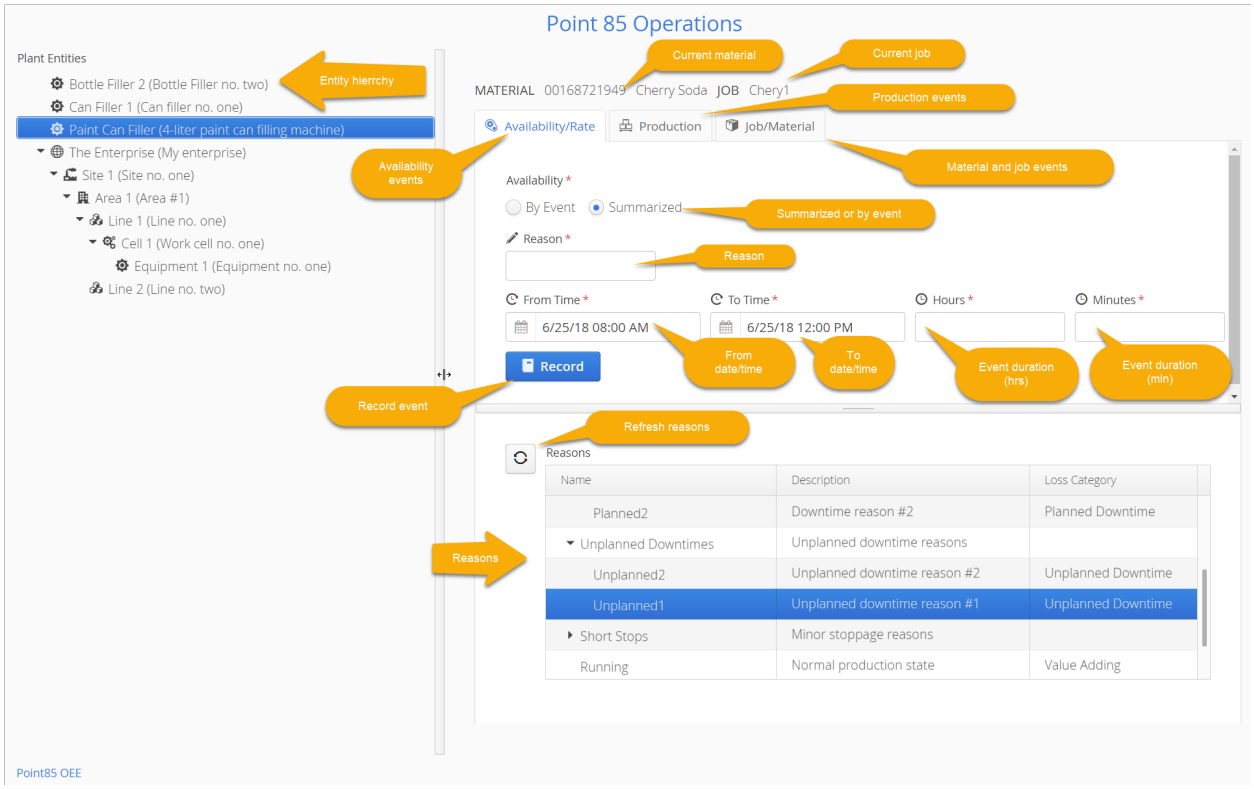
The monitor application has three main functions:

- Observe equipment performance via metrics available in the dashboard. For details on the dashboard, see the Dashboard section above.
- Observe notifications from the data collectors.
- Observe data collector status.

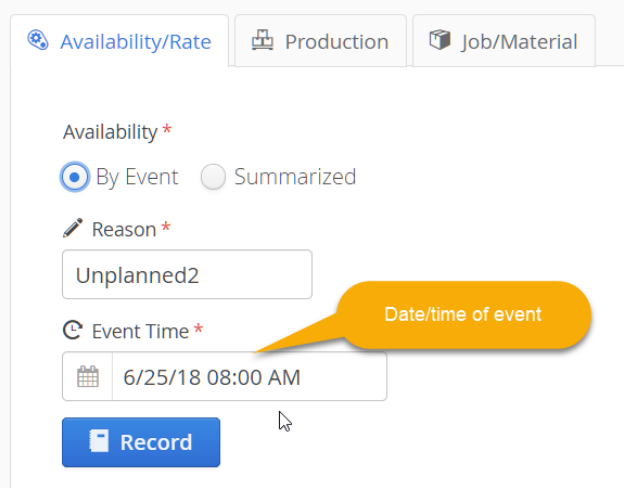
OPERATOR APPLICATION

The operator application is browser-based and allows a user to enter availability, performance, production, material change and job events. The events can be recorded in chronological order as they happened ("By Event") or in summary form ("Summarized") over a period of time.

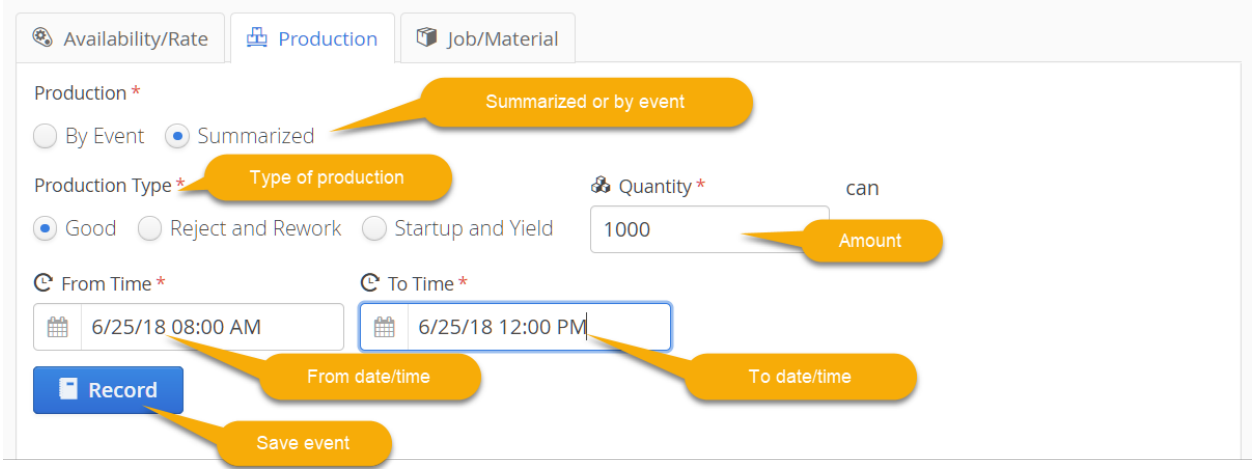
The user interface for availability/performance tab for a summarized event looks like:



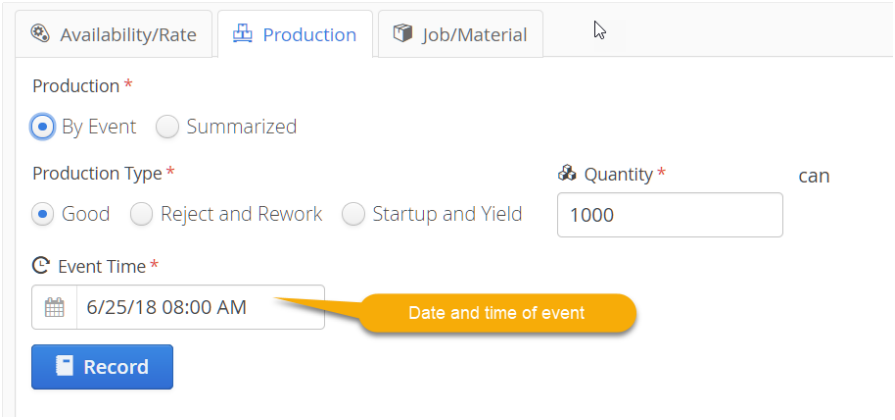
Note that if "By Event" is selected, the UI will change to allow entry of only the date and time of day when the availability event occurred:



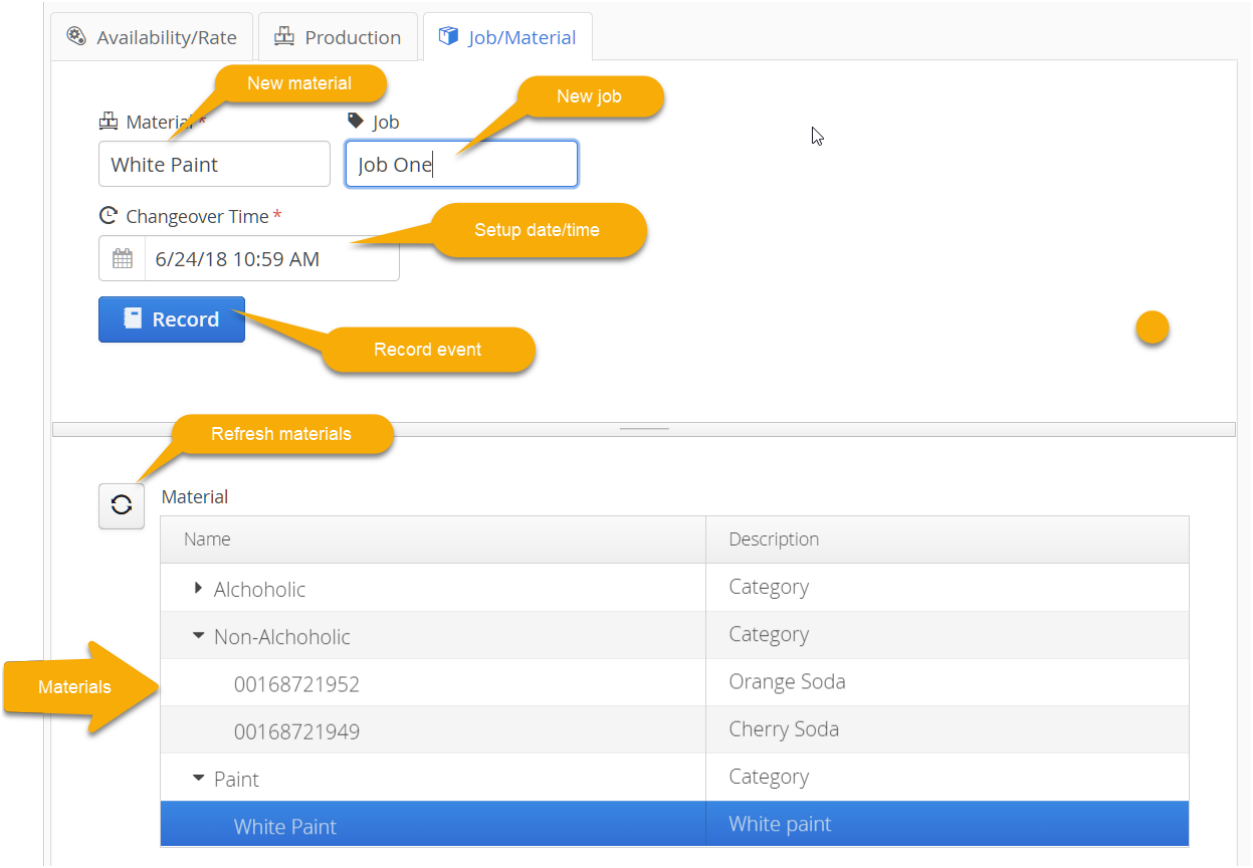
The production tab for a summarized event looks like:



Note that if “By Event” is selected, the UI will change to allow entry of only the date and time of day when the production event occurred:



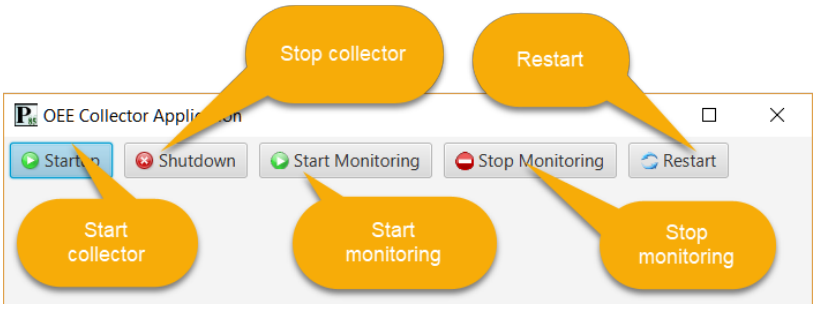
The material and job change tab looks like:



Testing Applications

Collector User Interface

This application provides a user interface for a data collector and is used for testing purposes. The application is launched from a host computer that has configured data sources. After the initial splash screen, the main form is displayed:



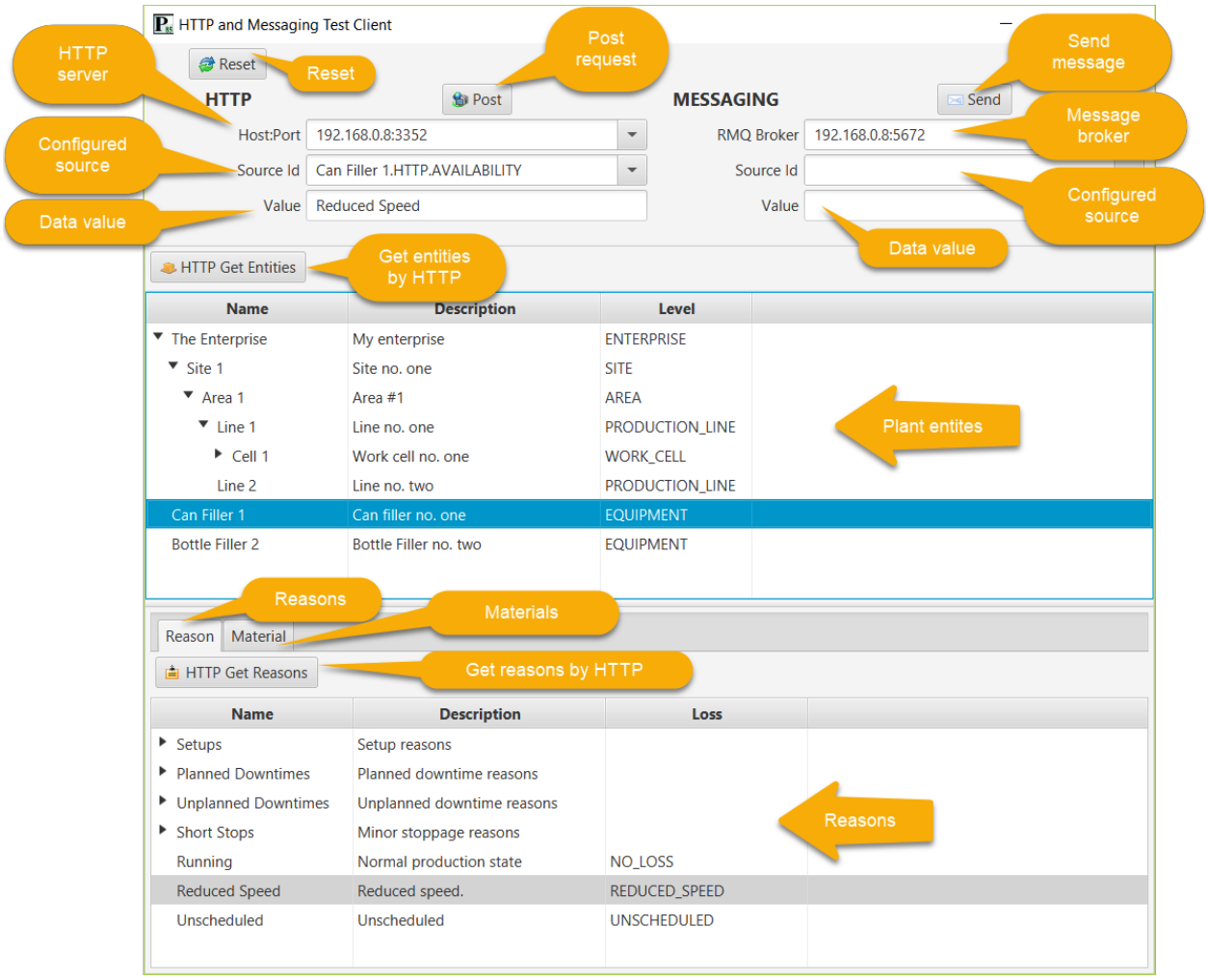
The actions are:

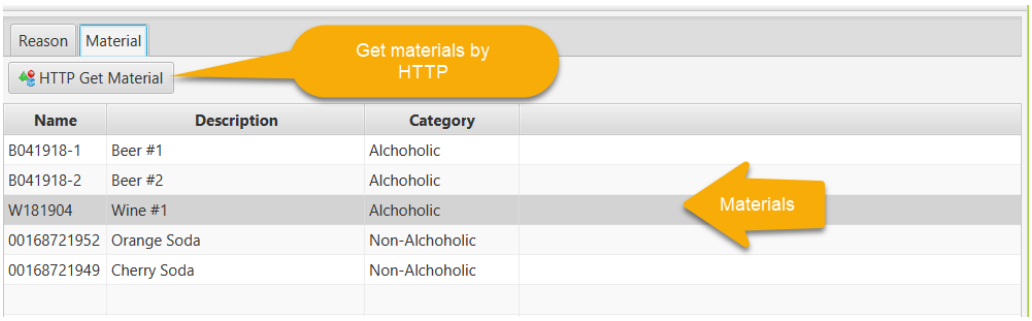
- *Startup*: Start the collector. It will connect to all OPC DA and OPC UA servers and be prepared to listen to HTTP requests as well as receive messages. It is in the monitoring state.
- *Shutdown*: Stop the collector. It is in the shutdown state.
- *Start Monitoring*: Start monitoring data input after having been stopped.
- *Stop Monitoring*: Stop monitoring all data inputs.
- *Restart*: Stop monitoring then restart monitoring.

HTTP AND MESSAGING

This application provides a user interface to send HTTP requests as well as RabbitMQ messages to a data collector configured with these data sources. It is used for testing purposes. It also exercises the HTTP APIs for fetching the plant entities, reasons and materials.

After the initial splash screen, the main form is displayed with the HTTP server and RabbitMQ broker comboboxes listing these data sources:





Clicking the *Reset* button re-queries the database for HTTP servers and RabbitMQ brokers and clears the source id and value fields.

HTTP

The configured HTTP servers will be listed in the combobox. Choose one to send requests to it. The actions are:

- *HTTP Get Entities*: get the plant entities and display them in the tree view. A GET request is made to the "entity" endpoint and PlantEntityResponseDto serialized object is returned. Selecting an equipment entity will populate the source id comboboxes with HTTP and messaging data sources.
- *HTTP Get Reasons*: get the reasons and display them in the tree table. A GET request is made to the "reason" endpoint and ReasonResponseDto serialized object is returned. Selecting a reason will populate the value text fields with the name of the reason.
- *HTTP Get Materials*: get the materials and display them in the table. A GET request is made to the "material" endpoint and MaterialResponseDto serialized object is returned. Selecting a material will populate the value text fields with the name of the material.
- *Post*: Make an HTTP POST equipement event request to the selected server with the event data.

Messaging

The configured RabbitMQ brokers will be listed in the combobox. Choose one to send messages to it. The actions are:

- Send: Send an equipment event message to the selected broker with the event data.

DATABASE

The Java Persistence 2.0 API (JPA) as implemented by the Hibernate ORM framework together with the Hikari connection pool is used to persist information to the database. The PeristenceService class is used for this purpose.

Hibernate and JPA abstract-away database specific aspects of inserting, updating, reading and deleting records in the tables. The API is designed to work with any relational database supported by Hibernate. In particular, Microsoft SQL Server 2012 dialect and Oracle ? have been tested.

DESIGN SCHEMA

PLANT_ENTITY Table

MATERIAL Table

REASON Table

EXECUTION SCHEMA

CONTRIBUTING TECHNOLOGY

https://github.com/eclipse/milo	Milo is an open-source implementation of OPC UA. It includes a high-performance stack (channels, serialization, data structures, security) as well as client and server SDKs built on top of the stack.
j-interop.org	j-Interop is a Java Open Source library (under LGPL) that implements the DCOM wire protocol (MSRPC) to enable development of Pure, Bi-Directional, Non-Native Java applications which can interoperate with any COM component.
openscada.org	openSCADA is an open source Supervisory Control And Data Acquisition System. It is platform independent and based on a modern system design that provides security and flexibility at the same time.
https://github.com/NanoHttpd/nanohttpd	NanoHTTPD is a light-weight HTTP server designed for embedding in other applications, released under a Modified BSD licence.
http://hibernate.org	Hibernate ORM enables developers to more easily write applications whose data outlives the application process. As an Object/Relational Mapping (ORM) framework, Hibernate is concerned with data persistence as it applies to relational databases (via JDBC).
https://brettwooldridge.github.io/HikariCP/	HikariCP is a “zero-overhead” production-quality connection pool.
https://www.rabbitmq.com/	RabbitMQ is an open source message broker software (sometimes called message-oriented middleware) that originally implemented the Advanced Message Queuing Protocol (AMQP) and has since been extended with a plug-in architecture to support Streaming Text Oriented Messaging Protocol (STOMP), Message Queuing Telemetry

	Transport (MQTT), and other protocols.
www.erlang.org	Erlang is a general-purpose, concurrent, functional programming language, as well as a garbage-collected runtime system.
https://mvnrepository.com/artifact/com.google.code.gson/gson	Gson JSON serializer and deserializer
https://vaadin.com/	Vaadin is an open-source platform for web application development.
https://wrapper.tanukisoftware.com	The Java Service Wrapper enables a Java Application to be run as a Windows Service or UNIX Daemon. It also monitors the health of your Application and JVM.
https://github.com/HanSolo/tilesfx	A JavaFX library containing tiles that can be used for dashboards.
https://www.bouncycastle.org	Bouncy Castle is a collection of APIs used in cryptography.
http://logging.apache.org/log4j/1.2/	Apache Log4j is a Java-based logging utility.
https://mvnrepository.com/artifact/org.slf4j	Log4j logging facade

REFERENCES

[Kennedy] *Understanding, Measuring, and Improving Overall Equipment Effectiveness.* How to Use OEE to Drive Significant Process Improvement, Ross Kenneth Kennedy, CRC Press, 2018.

[Stamatis] *The OEE Primer*, Understanding Overall Equipment Effectiveness, Reliability, and Maintainability, D.H. Stamatis, CRC Press, 2010.

[Koch] *OEE Industry Standard 2003*, Defining OEE for Optimal Loss Visualization, www.OEEFoundation.org, Arno Koch, 2003.

[Kraus] *OEE for Operators*, Overall Equipment Effectiveness, The Productivity Development Team, Productivity Press, 1999.