Using Apama Analytics Builder for Cumulocity IoT

Version 10.5

October 2019

Document ID: PAB-105-20191015

Table of Contents

About this Guide

This guide describes how to build and use models using Apama Analytics Builder for Cumulocity IoT.

Documentation Roadmap

This documentation is provided in the following formats:

- HTML
- PDF

In addition, the following is available which can be accessed from the model editor, which is one of the tools of Apama Analytics Builder for Cumulocity IoT:

- Reference information for the pre-built blocks: block descriptions, parameters, input port details, and output port details.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies service names and locations in the format *folder.subfolder.service* , APIs, Java classes, methods, properties.
Italic	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.

Convention	Description
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at "http://documentation.softwareag.com". The site requires credentials for Software AG's Product Support site Empower. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to "empower@softwareag.com" with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at "https://empower.softwareag.com/".

You can find product information on the Software AG Empower Product Support website at "https://empower.softwareag.com".

To submit feature/enhancement requests, get information about product availability, and download products, go to "Products".

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the "Knowledge Center".

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at "https://empower.softwareag.com/public_directory.asp" and give us a call.

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at "http://techcommunity.softwareag.com". You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.

- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

1 Release Notes

What's new in version 10.5

- You can now use Apama Analytics Builder with Cumulocity IoT Core, that is, with the version of Cumulocity IoT which is available in the cloud. With previous versions, it was only possible to use Apama Analytics Builder with Cumulocity IoT Edge, that is, with the local version of Cumulocity IoT.
- In the model editor, the hierarchical structure of devices and device groups is now reflected. This can be seen in the **Input** and **Output** categories of the palette, in the block parameter editor when selecting a different device, and when replacing devices.
    - When you use the search box in the palette of the model editor (available for input blocks and output blocks), all assets in the Cumulocity IoT inventory which match your search criteria are now shown (see "Adding a block" on page 43).
    - When you select a different device in the block parameter editor or when you replace devices, a dialog box is now shown in which you can select the required device or device group. When you use the search box in that dialog box, you can also select any other asset in the Cumulocity IoT inventory which matches your search criteria. See also "Editing the parameters of a block" on page 44 and "Replacing devices, device groups and assets" on page 50.
    - Using a tenant option, you can restrict the search to show only assets of a specify type, for example, to show only devices (see "Searching for input and output assets" on page 85).
- The following new block types are now available in the model editor:
    - **Alarm Input** - input block for processing received `Alarm` objects.
    - **Alarm Output** - output block for creating a new `Alarm` object.
    - **Event Output** - output block for creating a new `Event` object.
    - **Managed Object Input** - input block for processing received `ManagedObject` objects.
    - **Managed Object Output** - output block for updating a pre-configured property on the `ManagedObject` object.
    - **Operation Input** - input block for processing received `Operation` objects.

  See the Block Reference in the model editor for detailed information on the new block types.
- The input block types **Measurement Input** and **Event Input** are now able to ignore the timestamp of the incoming measurement or event. For this purpose, the **Ignore Timestamp** check box is now available in the block parameter editor. It is deselected (false) by default. See the Block Reference in the model editor for detailed information.

- Support for time-asynchronous output has been added for the output blocks. The output block type **Operation Output** is now able to produce asynchronous output. See also "Output blocks and event timing" on page 74.
- It is now possible to use multiple output blocks of type **Operation Output** in a single model or multiple models to create new operations for the same device.
- The **Expression** block can now call built-in static methods and access built-in constants for the supported input types. See the Block Reference in the model editor for detailed information.
- The **Threshold** block now supports the following additional options for the **Direction** parameter: **Above or Equal** and **Below or Equal**. See the Block Reference in the model editor for detailed information.
- In the model editor, it is now possible to use drag-and-drop to add a block from the palette to an expanded group on the canvas. Previously, when dragging a block from the palette, any groups on the canvas were ignored and the block was dropped on the canvas instead (below the group). See also "Adding a block" on page 43.
- Cumulocity IoT Edge is now configured in the same way as Cumulocity IoT Core. It is thus no longer possible to use configuration files in /usr/edge/properties/apama/extensions/config/files/; these have been removed. See also "Configuration" on page 95.
- You can now set or change various tenant options by sending REST requests to Cumulocity IoT. For detailed information, see "Configuration" on page 95.
- Status reporting is now disabled by default. See "Configuration" on page 95 for setting `status_period_secs` and `status_device_name`.
- The default name for the Cumulocity IoT device to which the status operations are to be published (`status_device_name`) has been changed from `c8y_EdgeGateway` to `apama_status`. See also "Keys for status reporting" on page 96.
- The default value for the concurrency level (`numWorkerThreads`) has been changed from 4 to 1. See also "Configuring the concurrency level" on page 80.
- It is no longer possible to use the Apama command-line tools to access the correlator.
- Support for creating extensions has been removed from Apama Analytics Builder. To create extensions, use the Apama Analytics Builder Block SDK instead.
- You can now use the Apama Analytics Builder Block SDK, which is available from "https://github.com/SoftwareAG/apama-analytics-builder-block-sdk", to write, test, and package custom blocks and to upload these blocks into Apama Analytics Builder. See also "Creating your own blocks" on page 27.

What's new in version 10.3.2

- Apama Analytics Builder now supports device groups that have been defined in the Cumulocity IoT inventory. They are available for selection from the **Input** category of the palette. See also “Adding a block” on page 43.
- When you expand the **Input** or **Output** category in the palette, a search box is now shown that you can use to list the devices and device groups that you are looking for. The search is case-sensitive. For more information, see “Adding a block” on page 43.

  > **Note:** Apama Analytics Builder only supports devices that are marked with a `c8y_IsDevice` fragment in the Cumulocity IoT inventory. Child devices are not supported.

- The number of devices (and device groups) that are shown in the palette can now be customized. For detailed information, see “Configuring the number of shown devices, device groups and/or assets” on page 84.
- A special output block, the **Trigger Device**, can now be selected from the palette; it sends the output back to the device which triggered the output. Models can now process data from multiple devices, and scale up (using multiple cores) when doing so. For detailed information, see “Model execution for different devices” on page 78.
- Apama Analytics Builder now supports input devices that are referred to as "broadcast devices". Signals from these devices are available to all models across all devices. For detailed information, see “Broadcast devices” on page 80.
- In previous versions of Apama Analytics Builder, it was possible to create a model that received data from multiple devices or sent data to multiple devices. This is no longer possible in the default configuration; trying to activate such a model will fail with an error. In order to run such models, the concurrency level must be set to 1 (see “Model execution for different devices” on page 78). This will limit the performance of executing analytics.
- On the canvas of the model editor, the labels of the following blocks now include the value of the most important parameter:
  - The **Expression** block now shows the defined expression.
  - The **Threshold** block now shows the defined threshold value.
  - The **Combiner** block now shows the defined mode.
  - The **Extract Property** block now shows the defined property name.
  - The **Rounding** block now shows the defined rule.
  - The **Time Delay** block now shows the defined delay in seconds.

- The **Expression** block now handles Boolean literals and logical operators in a case insensitive manner. This is different from Apama's Event Processing Language (EPL) which only accepts lowercase Boolean literals and logical operators.
- The toolbar button for leaving the model editor is now located at the very right of the toolbar. See also "Leaving the model editor" on page 42.
- It is now possible to use the Apama command-line tools to monitor aspects of the correlator, including queue sizes and access to the log file.
- It is now possible to customize Apama Analytics Builder by using extensions.
- The following paths are now used (now including extensions/config/files in the path):
  - /usr/edge/properties/apama/extensions/config/files/support/cumulocity/
  - /usr/edge/properties/apama/extensions/config/files/framework/monitors/

  Note: The location of the configuration files may be subject to change in future major releases.
- A new topic with information on virtual devices has been added to the documentation. See "Virtual devices" on page 81.

What's new in version 10.3.1

- The **Expression** block now also supports the `string` and `boolean` types both for input and result values. Logical operators can now be used on Boolean values. Support for the existing relational, numerical and equality operators has been extended to the `string` and `boolean` types where appropriate. See the Block Reference in the model editor for detailed information on the new functionality.
- The **Crossing Counter** block can now operate over a time-bounded window that is specified with the **Window Duration** parameter. See the Block Reference in the model editor for more information on the new functionality.
- A new **Standard Deviation** block is now available. This block can be used to calculate the standard deviation and variance of the values over time. See the Block Reference in the model editor for detailed information on this block.
- A new **Combiner** block is now available. This block can be used to calculate the output based on the selected mode (such as maximum or latest) and the connected inputs. See the Block Reference in the model editor for detailed information on this block.
- A new **Gradient** block is now available. This block can be used to calculate the weighted linear regression gradient for the values. See the Block Reference in the model editor for detailed information on this block.

- The blocks, wires and groups on the canvas now always snap to a grid. You can decide whether the grid is to be shown or not. It is not shown by default. See also "Showing and hiding the grid" on page 59.
- A German user interface is now available for Apama Analytics Builder. See also "Language settings" on page 20.

2 Getting Started with Apama Analytics Builder

What is Apama Analytics Builder?

Apama Analytics Builder runs as an application in Cumulocity IoT, which is a web-based platform for managing IoT devices (see also "https://www.cumulocity.com/"). It allows you to build analytic models that transform or analyze streaming data in order to generate new data or output events. The models are capable of processing data in real time.

You build the models in a graphical environment by combining pre-built *blocks* into *models*. The blocks in a model package up small bits of logic, and have a number of inputs, outputs and parameters. Each block implements a specific piece of functionality, such as receiving data from a sensor, performing a calculation, detecting a condition, or generating an output signal. You define the configuration of the blocks and connect the blocks using *wires*. You can edit the models, simulate deployment with historic data, or run them against live systems. See "Understanding Models" on page 21 for more detailed information.

Apama Analytics Builder consists of the following tools:

- **Model manager.** When you invoke Apama Analytics Builder, the model manager is shown first. It lists all available models and lets you manage them. For example, you can test and deploy the models from the model manager, or you can duplicate or delete them. You can also create new models or edit existing models; in this case, the model editor is invoked. See "Using the Model Manager" on page 31 for detailed information.
- **Model editor.** The model editor lets you define the blocks that are used within a model and how they are wired together. User-visible documentation (the so-called *Block Reference*) is available in the model editor, describing the functionality of each block. See "Using the Model Editor" on page 39 for detailed information.

The blocks are implemented in the Event Processing Language (EPL) of Apama. At runtime, the EPL code runs in an Apama correlator to execute the models. Some runtime behavior and restrictions are important to understand. These are documented in later chapters.

Apama Analytics Builder and Cumulocity IoT

Cumulocity IoT is a platform for connecting, monitoring and controlling remote devices. For an overview, see the *Concepts guide* at "https://www.cumulocity.com/guides/". Apama Analytics Builder runs as a component within the Cumulocity IoT platform.

Devices and sensors can be connected to Cumulocity IoT. See the information on interfacing devices in the *Concepts guide* and the information on device integration using MQTT in the *Device SDK guide*, both available at the above URL.

Sensors result in `Measurement` or `Event` objects in Cumulocity IoT, and devices can receive `Operation` objects created within the Cumulocity IoT platform. All of these objects (`Measurement`, `Event`, `Operation`) will be associated with a single device in the Cumulocity IoT platform. A device may have multiple types of measurement associated with it, and the types of measurements each device supports may be the same as other devices or different to other devices. Once devices are connected to Cumulocity IoT, information about these devices is stored in the Cumulocity IoT inventory. These are visible in the Device Management application, which can also be used to view `Measurement`, `Event` or `Operation` objects associated with that device. See the information on device management in the *User guide*, available at the above URL.

The Cumulocity IoT platform includes an Apama correlator component, which is managed by the Cumulocity IoT platform (this is not manually started or stopped) and is preconfigured to communicate to Cumulocity IoT. This correlator hosts the Apama Analytics Builder runtime, and also executes any custom Apama rules added using the Apama EPL Apps web application. For more information, see the *Streaming analytics guide*, available at the above URL.

The Apama Analytics Builder web application is available via the application switcher after logging in to the Cumulocity IoT web interface. See also "Starting Apama Analytics Builder" on page 20.

Apama Analytics Builder allows you to create models that interact with the devices and sensor measurements. Models can receive `Measurement` and `Event` objects from devices, which provide the inputs to calculations or pattern detection performed within a model. Models can create new `Measurement` objects which can represent derived values from sensors (for example, an average temperature) or the measurements can be used as an input to other analytic models (see "Connections between models" on page 82). Models can create new `Operation` objects which are sent to devices to control the devices (for example, to sound an alarm bell, display a message on a screen, or switch a device off). The models are also stored in the Cumulocity IoT inventory, but can be imported or exported via the model manager.

Business logic can also be written in Apama's Event Processing Language (Apama EPL) which gives more power and flexibility in a text-based programming language. This is an alternative if more complex logic is required or the logic does not fit into the pattern of an Apama Analytics Builder model. Apama EPL applications can be written directly in Cumulocity IoT using the Apama EPL Apps web application. See the *Streaming analytics guide* at "https://www.cumulocity.com/guides/" for more information, including examples. Alternatively, it is also possible to build custom blocks if none of the blocks delivered with Apama Analytics Builder implement the logic required; see "Creating your own blocks" on page 27.

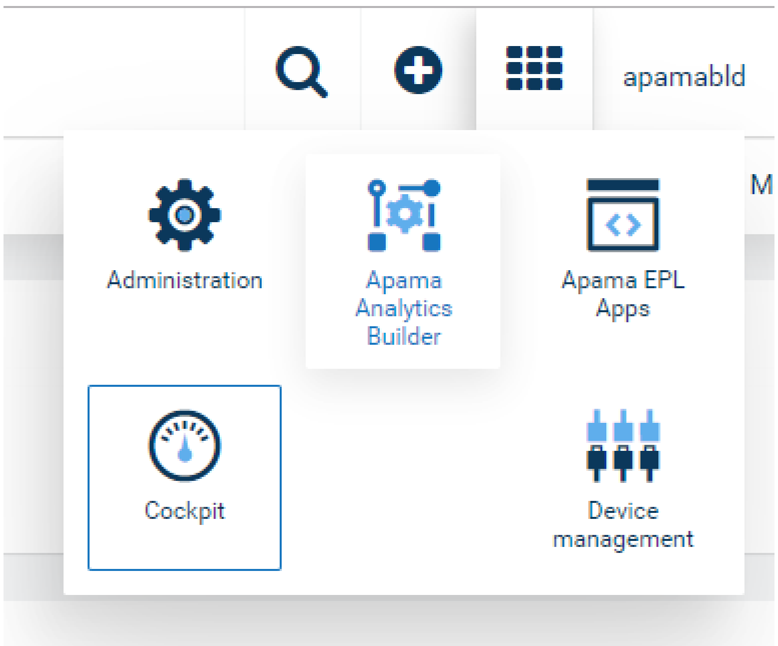
Apama Analytics Builder can be used with both Cumulocity IoT Core (cloud) and Cumulocity IoT Edge (local installation). You can customize several aspects of Apama Analytics Builder by setting various tenant options. See "Configuration" on page 95 for detailed information.

Prerequisites

Apama Analytics Builder supports the same browsers as Cumulocity IoT, with the following exception: browsers on smartphones and tablets are not supported.

Starting Apama Analytics Builder

Apama Analytics Builder can be accessed from Cumulocity IoT, using the application switcher which is available in the top bar of the Cumulocity IoT user interface.



For detailed information on how to use Cumulocity IoT, see "https://www.cumulocity.com/guides/".

Ask your administrator for the URL that is required to start Cumulocity IoT.

Language settings

The language in which the user interface of Apama Analytics Builder is shown depends on your user settings in Cumulocity IoT. See the *User guide* at "https://www.cumulocity.com/guides/" for more information.

Apama Analytics Builder currently supports English and German, even though Cumulocity IoT supports further languages. If Cumulocity IoT or the browser is set to a language that is currently not supported by Apama Analytics Builder, the user interface is shown with the default language, which is English.

3 Understanding Models

Models

A model is a container which can have a network of blocks connected to each other with wires.

The behavior of a block inside a model does not depend on other blocks. There can be multiple instances of the same block in a model where each instance may behave differently, depending on the configurable parameters or the inputs connected to the block.

Blocks

Blocks are the basic processing units of the model. Each block has some predefined functionality and processes data accordingly. A block can have a set of parameters and a set of input ports and output ports.

The palette of the model editor offers for selection the following types of blocks:

- Input blocks, which receive data from external sources. An input block normally represents a device or device group that has been registered in the Cumulocity IoT inventory. See also “Input blocks” on page 23.
- Output blocks, which send data to external sources. An output block normally represents a device that has been registered in the Cumulocity IoT inventory. See also “Output blocks” on page 23.
- Processing blocks, which receive data from the input blocks and send the resulting data to the output blocks. See also “Processing blocks” on page 23.

A block can receive data from another block through its input ports. A block can send data to another block through its output ports. Different blocks will have different numbers of input or output ports, and some blocks have only input ports or only output ports. For most blocks, it is not required to connect all of the input or output ports.

A block can have configurable parameters that define the behavior of the block. These parameters are either optional or mandatory, depending on the requirement of the block.

When using the same block multiple times, you can specify different values for the same parameter. For example, the **Threshold** block has a configurable parameter named **Threshold Value**. If you are using two instances of the **Threshold** block and configure this parameter differently for each block, the blocks will report different breaches of the threshold.

Note: Two output ports cannot be connected to same input port, whereas one output port can be connected to multiple input ports.

Input blocks

An input block is a special type of block that receives data from an external source. It converts the data into a format understandable to wires and transfers the data to the connected blocks. For example, when an input block receives a `Measurement` event from Cumulocity IoT, it extracts the required information from the event and then transfers the information to the connected blocks for further processing.

Models can process data from multiple devices, and scale up (using multiple cores) when doing so. For detailed information, see "Model execution for different devices" on page 78.

In addition, Apama Analytics Builder supports input devices that are referred to as "broadcast devices". Signals from these devices are available to all models across all devices. For detailed information, see "Broadcast devices" on page 80.

Output blocks

An output block is a special type of block that receives data from a connected processing block. It converts the data into a format understandable to an external source and transfers the data to the external source. For example, when an output block receives data from a connected processing block, it packages the data into an `Operation` object and then sends the operation to Cumulocity IoT.

The **Trigger Device** is a special output block which can be used to send the output back to the device which triggered the output. Models can process data from multiple devices, and scale up (using multiple cores) when doing so. For detailed information, see "Model execution for different devices" on page 78.

Processing blocks

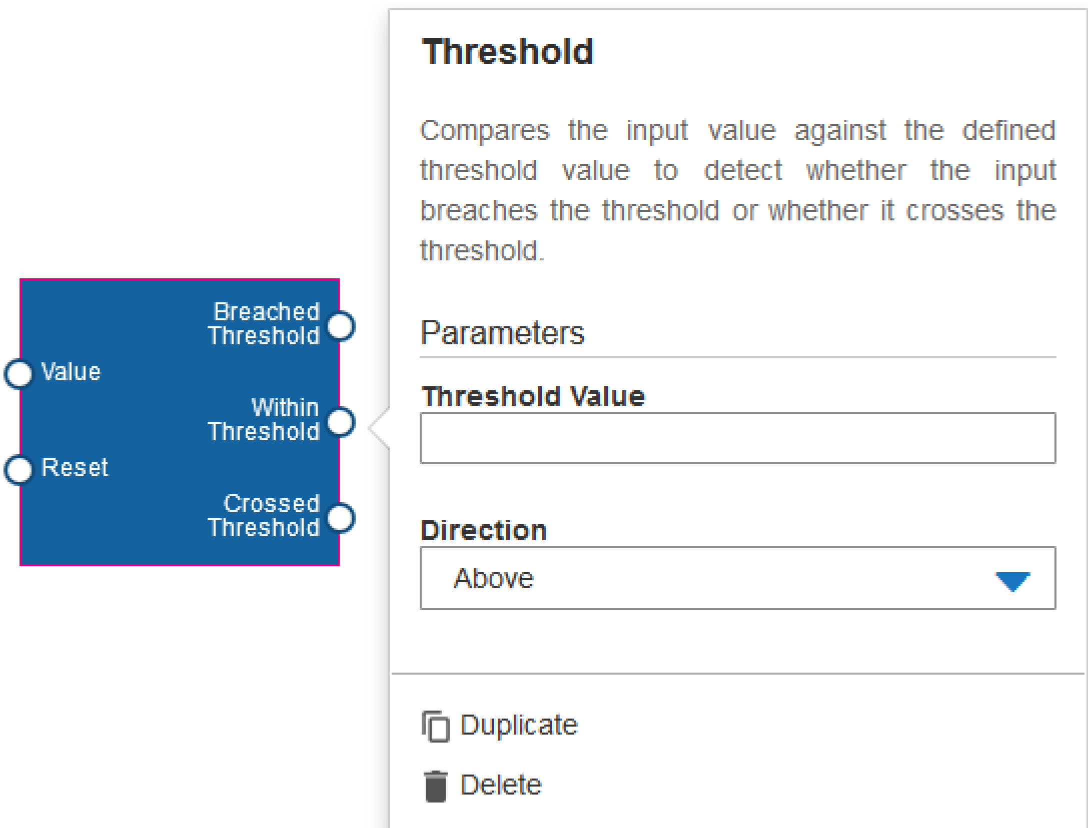
There are different types of processing blocks. They are grouped into different categories in the palette in the model editor, depending on their functionality.

This category	includes blocks that
Logic	perform logical operations on the data. Blocks such as **AND** and **OR** are in this category.
Calculation	perform mathematical operations on the data. Blocks such as **Difference**, **Threshold**, **Direction Detection**, **Delta** and **Expression** are in this category.

This category	includes blocks that
Aggregate	perform aggregation of the data over a window of values. Blocks such as **Average (Mean)** and **Integral** are in this category.
Flow Manipulation	manipulate the flow of the data. Blocks such as **Time Delay**, **Gate**, **Pulse** and **Latch Values** are in this category.
Utility	provide miscellaneous utility functions. Blocks such as **Toggle** and **Missing Data** are in this category.

Example of a processing block - the Threshold block

The following example shows what a block looks like in the model editor, together with the block parameter editor. It shows the **Threshold** block, which detects whether the input value breaches the threshold or whether it crosses the threshold.



The parameters are:

- **Threshold Value**. `float` type. This value is compared against the input value.

- **Direction**. The direction in which to look: whether the input value is above or below the defined threshold, or whether it crosses the threshold.

The input ports are:

- **Value**. `float` type. The input value to the block, to be compared against the defined threshold value.
- **Reset**. `pulse` type. When a signal is received, the state of the block is reset so that any previously received input values are no longer used.

The output ports are:

- **Breached Threshold**. `boolean` type. Is set to `true` when the threshold has been breached. That is, the input value is beyond the range of the defined threshold value.
- **Within Threshold**. `boolean` type. Is set to `true` when the threshold has not been breached. That is, the input value is within the range of the defined threshold value.
- **Crossed Threshold**. `pulse` type. Sends a signal when the input value crosses the threshold, going from one side of the threshold to the other.

Overview of all blocks

The following table gives a brief description of all blocks that can be selected from the palette of the model editor, sorted alphabetically.

Block name	Description
AND	Performs a logical 'and' on the inputs.
Average (Mean)	Calculates the mean of the values over time.
Combiner	Calculates the output based on the selected mode and the connected inputs.
Crossing Counter	Detects and counts the number of threshold crossings in the specified direction.
Delta	Calculates the difference between successive input values.
Difference	Calculates the absolute and signed differences between the connected inputs.
Direction Detection	Detects whether the input value changes direction.

Block name	Description
Expression	Evaluates an expression to perform arithmetic or logical calculations or string operations.
Extract Property	Extracts the specified property from the input value and converts it to the specified type.
Gate	Blocks the input from going to output unless the gate is open and enabled.
Gradient	Calculates the weighted linear regression gradient for the values.
Input	Input block for measurements or other inputs from a device or device group.
Integral	Calculates the integral of the input value over time.
Latch Values	Latches the latest input value received while the block is enabled.
Minimum / Maximum	Calculates the minimum and maximum of a value over time.
Missing Data	Generates an output if the input has not occurred for a set amount of time.
NOT	Performs a logical 'not' on the input.
OR	Performs a logical 'or' on the inputs.
Output	Output block for creating measurements or other outputs for a device (or the triggering device).
Pulse	Converts a non-pulse input into a pulse output.
Range Lookup	Finds the range in which the input value lies.
Rounding	Rounds the input to a specified number of decimal points or to an integer, using a selectable rule.
Standard Deviation	Calculates the standard deviation and variance of the values over time.

Block name	Description
Threshold	Compares the input value against the defined threshold value to detect whether the input breaches the threshold or whether it crosses the threshold.
Time Delay	Delays the input by the specified amount of time.
Toggle	Converts two pulse inputs to a boolean output based on the set and reset signals, with optional delays.

Creating your own blocks

You can use the Apama Analytics Builder Block SDK to write, test, and package custom blocks and to upload these blocks into Apama Analytics Builder.

The Block SDK is available from GitHub at "https://github.com/SoftwareAG/apama-analytics-builder-block-sdk". See the documentation in GitHub for detailed information.

You write the custom blocks in Apama's Event Processing Language (EPL). Once you have written a block, you can package it into an *extension* and upload it. An example command line to build and upload an extension is:

```
analytics_builder build extension --input path --cumulocity_url $C8Y_URL
--username $C8Y_USERNAME --password $C8Y_PASSWORD --name customBlocks
```

Wires

One block is connected to another block with the help of wires. All data transfer between the output port of one block and the input port of another block is done using wires. All connections must be made between compatible types. See "Wires and Blocks" on page 61 for detailed information.

Note: The network of blocks in a model cannot contain any kind of cycles. See "Wire restrictions" on page 71 for more information.

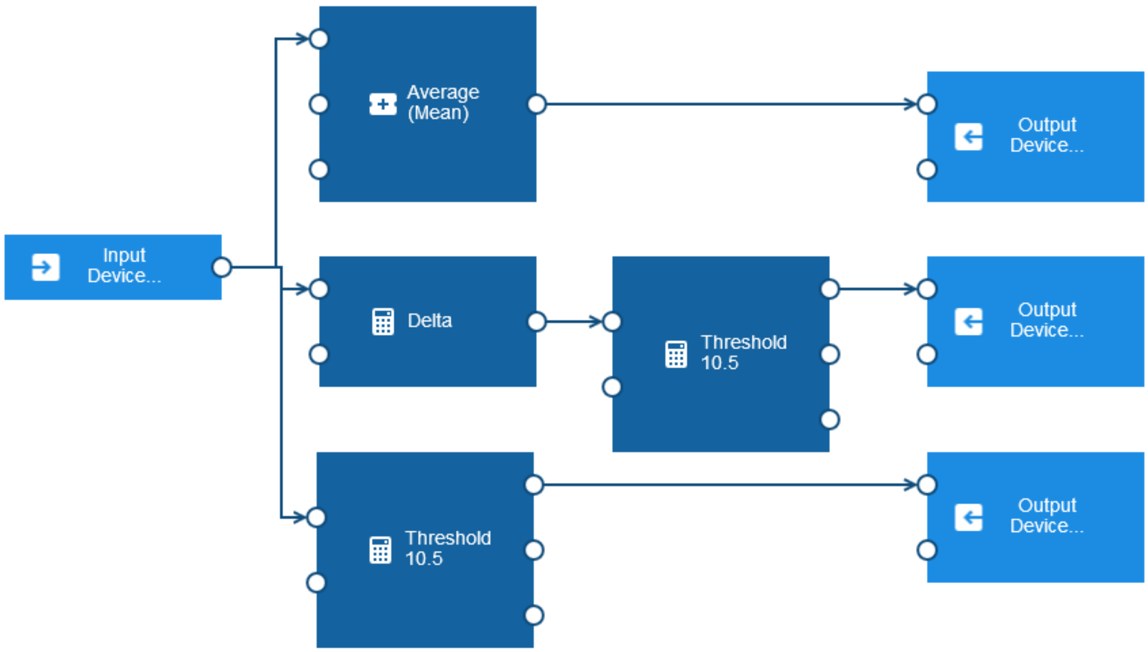
Sample use case

Consider a situation where you are getting real-time sensor data and you want to analyze this data. For the sake of simplicity, let us assume that there is only one sensor and that you are interested in the following:

- You want to know the average value of the sensor readings over a period of time.

- You want to detect sudden changes in the sensor readings using a defined threshold value.
- You want to ensure that the sensor readings are within a certain range and that an alert is created if the readings go beyond that range. For example, you are getting pressure readings and you want to ensure that the maximum pressure does not go beyond the range that the device can handle.

The model for this example has the following blocks:



- An input block which shows **Input Device** as the device name

  The incoming data is in real time and continuous. The input block receives the data from the sensor. It passes the data to the **Average (Mean)**, **Delta** and **Threshold** blocks. The input ports of these blocks are connected to the output port of the input block.
- An **Average (Mean)** block

  This block finds the average (or mean) of the readings that it receives over a period of time and passes this to the connected output block.
- A **Delta** block

  This block calculates the difference between successive input values and passes the calculated value to the connected **Threshold** block.
- Two different instances of a **Threshold** block

  A **Threshold** block compares the input value against the defined threshold value to detect whether the input breaches the threshold or not.

  The first instance is connected to the **Delta** block and reports a breach if the delta value goes beyond the threshold.

The second instance is connected to the input block and reports a breach if the input value is not within the threshold.

- Three instances of an output block which show **Output Device** as the device name

  The first instance sends the average of the sensor reading.

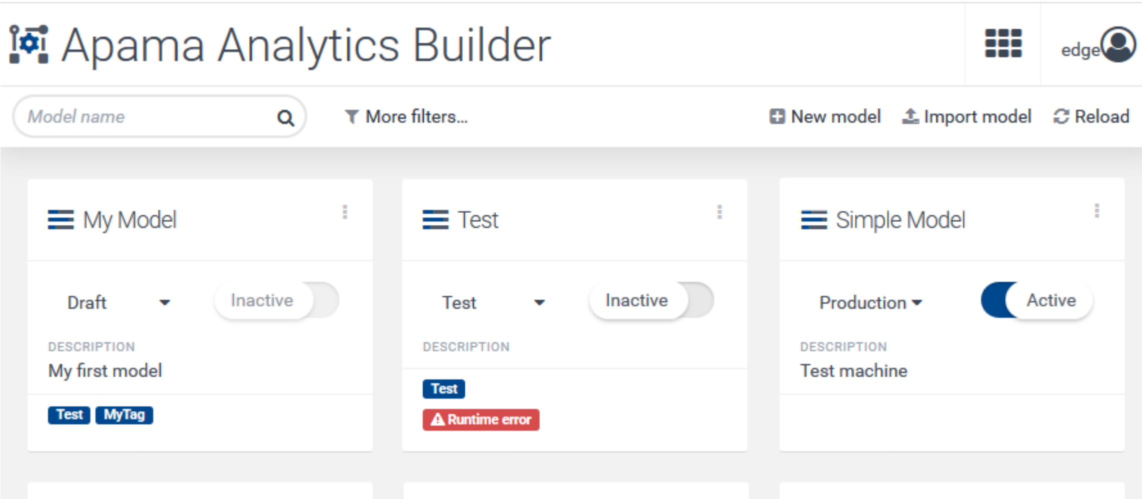
  The second instance generates an output if the values of successive sensor readings change by more than the configured threshold.

  The third instance generates an output if the sensor value goes beyond the configured threshold.

4 Using the Model Manager

The model manager user interface

The model manager lists all available analytic models within the current Cumulocity IoT environment as cards. Each card shows the mode (such as **Draft** or **Production**) and state (**Active** or **Inactive**) of a model. You add new models and manage the existing models from here.



Each card that is shown for a model has an actions menu (the three vertical dots that are shown at the top right of a card) which contains commands for managing the model (for example, to export or remove the model).

To edit a model, you can simply click on the card that is shown for the model (see also "Editing an existing model" on page 35). When you add a new model or edit an existing model, the model editor is invoked in which you define the blocks and wires that make up a model. See "Using the Model Editor" on page 39 for detailed information.

If one or more tags have been defined for a model, they are shown on the card for that model.

If you want to change the name, the description or the tags of an existing model, you have to do this in the model editor. See "Changing the name, description, and tags of a model" on page 41.

If you have a long list of cards, you can easily locate the model that you are looking for by entering its name in the **Model name** search box. Or you can enter part of the model name. For example, enter the word "test" to find all models that have this word in their names. The characters that you type in may be contained at any position within the model name. These search criteria are not case-sensitive. When search criteria are currently applied, an ✖ is shown next to the search box; click this to clear the search and thus to show all available cards.

You can also reduce the number of shown cards by using a filter. See “Filtering the models” on page 33 for detailed information.

If **Runtime error** is shown on the card of a deployed model, this model is no longer processing events. Move the mouse pointer over this message to display a tooltip with information on what went wrong.

See the Cumulocity IoT documentation for detailed information on the items that are shown in the top bar (application switcher and user button).

Filtering the models

The model manager offers several ways to reduce the number of shown cards, thus letting you quickly locate the models that you are looking for.

Filtering also works in combination with a model name that you specify in the **Model name** search box which is explained in “The model manager user interface” on page 32.

To filter the models

1. In the toolbar of the model manager, click **More filters**.
2. In the resulting dialog, select one or more filters.

   You can filter the models according to the following criteria:

   - **Mode**

     You can show only the models that are in a specific mode. For example, if you only want to see the models that are in simulation and test mode, select the corresponding check boxes.

   - **Status**

     You can show only the models that are in either the Active or Inactive state. For example, if you only want to see active models, select the corresponding check box.

   - **Device or device group**

     You can show only the models that use specific devices in their input blocks and output blocks, specific device groups in their input blocks, or even specific assets. Open the **Filter by device or device group name** drop-down list box, select one or more devices, device groups, and/or assets and click **Apply**.

   - **Data point**

     You can show only the models that use specific data points, such as `c8y_TemperatureMeasurement`. This requires that at least one device has been selected in the **Filter by device or device group name** drop-down list box. Open the **Filter by data points** drop-down list box, select one or more data points, and click **Apply**.

- **Tags**

  You can show only the models for which specific tags have been defined in the Model Configuration dialog box, which is shown when you add a new model or when you invoke that dialog box from the model editor (see also "Adding a new model" on page 34 and "Changing the name, description, and tags of a model" on page 41). Open the **Filter by tag** drop-down list box, select one or more tags, and click **Apply**.

All of the above-mentioned drop-down list boxes include a **Filter** search box that you can use to reduce the number of items that are offered for selection. You can enter a name or part of a name. For example, enter the word "test" to show only the items that have this word in their names. The characters that you type in may be contained at any position within the name. These filter criteria are not case-sensitive. Clicking the **All** check box selects all items that are currently shown in the drop-down list box, depending on the contents of the **Filter** search box.

You can combine several types of filters, for example, to show only active models in production mode that use a specific device.

3. Click **Apply filters**.

   The toolbar of the model manager now shows the types of filters that are currently applied. For example:

   Click **Clear filters** in the toolbar if you want to clear these filters. Or to clear a specific filter, click on a filter icon in the toolbar, and deselect the filter in the resulting dialog box. Clicking **Reset filters** in that dialog box also clears the filters.

Adding a new model

When you add a new model, the model editor is invoked. See "Using the Model Editor" on page 39 for detailed information.

Note: The new model will only be listed in the model manager, when you save the model in the model editor. See also "Saving a model" on page 42.

To add a new model

1. In the toolbar of the model manager, click **New model**.
2. In the resulting Model Configuration dialog box, enter a unique model name.

   You can optionally enter a description for the model and one or more tags.

   Tags are helpful for filtering the models in the model manager to show only the models for which a specific tag has been defined (see also "Filtering the models" on page 33). To add a tag, you simply type its name and press Enter or the Tab key.

The tag is then shown in a colored rectangle. To remove a tag, click on the that is shown in the rectangle. The dialog prevents you from entering duplicate tags for a model; if you enter such a tag name, the duplicate tag is not added and the original tag blinks one time.

3. Click **OK**.

   The model editor appears. See "Overview of steps for adding a model" on page 41 for a brief overview of how to add blocks and wires to the new model.

Editing an existing model

You can edit each model that is currently listed in the model manager.

If a model is in the Active state, editing will set the model to read-only mode. In this case, the model editor only allows you to view the contents of the model (for example, you can view the block parameters). You can navigate and zoom the model as usual, but you cannot change anything. The save button in the model editor is therefore disabled.

To edit a model

- In the model manager, click the actions menu of the model that you want to edit (or view) and then click **Edit**.

  Or simply click the card that is shown for the model (but not on the toggle button for changing the state or the drop-down menu for changing the mode).

  If the model is in the Active state, a dialog appears informing you that you can only view the model. When you click **Continue**, the model editor appears and you can view the model, but you cannot change it.

  See "Using the Model Editor" on page 39 for further information.

Deploying a model

A model can have one of two states. The current state is always indicated on the toggle button that is shown for a model:

- **Active.** This state indicates that the model has been deployed.
- **Inactive.** This state indicates that the model is currently not deployed.

The inputs that a model receives and what happens to its outputs depends on the mode to which the model is set. Each model can be set to one of the following modes:

- **Draft.** The model is still under development. (New models are created in draft mode.)
- **Test.** This mode is only permitted for models using a single device. When active, the model is deployed to the Apama correlator so that the measurements and events from the device are processed. The output of the model is only stored (and recorded

as an `Operation` or `Measurement` object of a "virtual device") and *not* sent back to the device.

- **Simulation.** This mode is only permitted for models using a single device. When active, the model uses historical input data (replayed in real time from previously received data) and is deployed to the Apama correlator. The output of the model is only stored (and recorded as an `Operation` or `Measurement` object of a "virtual device") and *not* sent back to the device. To start a simulation, you must define the time range from which the input data is to be used. When all data from the time range has been replayed, the model is automatically undeployed from Apama and the model state is changed to Inactive. The timestamps of the historical data entries remain unchanged for easier comparison of simulation runs. See also "Model Simulation" on page 87.
- **Production.** When active, the model is deployed to the Apama correlator so that the measurements and events from the devices are processed. The output of the model is stored and sent back to the devices.

A model in draft mode can only be in the Inactive state. A model in test, simulation or production mode can be in either the Active or Inactive state.

When a model is imported by loading a JSON file, it is always imported as an inactive model.

To deploy a model

1. In the model manager, click the drop-down menu in the model that you want to deploy and select one of **Production**, **Test** or **Simulation**.
2. If you have selected simulation mode, click the calendar icon which is now shown, specify the time span that is to be used, and click **Apply**. See also "Simulation parameters" on page 88.
3. When the toggle button currently shows **Inactive**, click this button to change the state to **Active**. For simulation mode, you can only set the state to **Active** when a valid time range has been defined.

Undeploying a model

You can undeploy each model that is currently in production, test or simulation mode and for which the toggle button shows **Active**.

When you undeploy a model, the model is stopped and no longer processes incoming data. Any state built up in the model is lost. For simulation mode, this means that the model is stopped before all historical data from the specified time range has been replayed.

To undeploy a model

- In the model manager, click the toggle button in the model that you want to undeploy so that **Inactive** is then shown on the button.

Copying a model

You can copy each model that is currently listed in the model manager.

When you copy a model, the copy gets the same name as the original model followed by the number sign (#) and a number. For example, when the name of the original model is "My Model", the name of the first copy is "My Model #1". The number in the model name is increased by one with each subsequent copy that you create. The copy gets the same description as the original model. It is recommended that you edit the copy and give the model a meaningful name and description.

To copy a model

- In the model manager, click the actions menu of the model that you want to copy and then click **Copy**.

  A card for the copied model is immediately shown in the model manager.

Exporting a model

You can export each model that is currently listed in the model manager. This is helpful, for example, if you want to transfer a model from the current Cumulocity IoT tenant to a different tenant. The model is saved in JSON format.

To export a model

- In the model manager, click the actions menu of the model that you want to export and then click **Export**.

  The resulting behavior depends on your browser. The model is usually exported to the download location of your browser.

Importing a model

You can import a model that has previously been exported (in JSON format). This is helpful, for example, if you want to import a model from a different Cumulocity IoT tenant.

To import a model

1. In the toolbar of the model manager, click **Import model**.
2. In the resulting dialog box, navigate to the location where the model that you want to import is stored.
3. Select the model and click **Open**.

   A card for the imported model is shown in the model manager.

Removing a model

You can remove each model that is currently listed in the model manager. When you remove a model that is currently deployed, it is first undeployed and then removed.

To remove a model

1. In the model manager, click the actions menu of the model that you want to remove and then click **Remove**.
2. In the resulting dialog box, click **Remove** to confirm the removal.

Reloading all models

You can refresh the display to show any changes other users have made since the page loaded, or to see whether deployed models have entered a failed state.

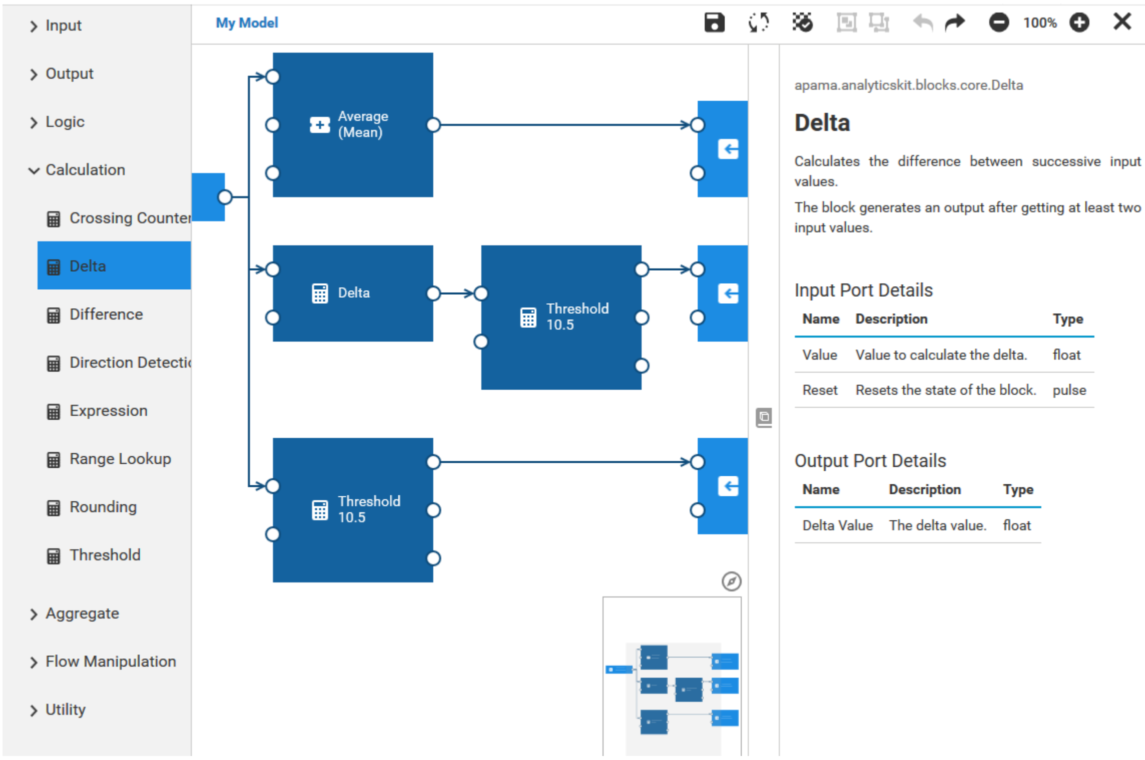
To reload all models

- In the toolbar of the model manager, click **Reload**.

5 Using the Model Editor

The model editor user interface

The model editor allows you to create analytic models graphically. It is invoked when you add or edit a model in the model manager. See also “Adding a new model” on page 34 and “Editing an existing model” on page 35.



The palette on the left contains the blocks that you can add to your model. It has several expandable/collapsible categories for the different types of blocks.

The canvas in the middle is the area in which you "draw" your model. You drag the blocks from the palette onto the canvas, specify the parameters for the blocks, and wire the blocks together. The content of the canvas is aligned to a grid (see also “Showing and hiding the grid” on page 59).

The overview area at bottom right of the canvas shows the entire model. This is helpful if your model is too large to fit on the currently visible area of the canvas. See also “Navigating large models” on page 57.

The documentation pane on the right allows you to view reference information for the currently selected block. See also “Viewing the documentation for a block” on page 46.

Caution: Changes are only saved when you click (see also “Saving a model” on page 42). The editor warns you if you attempt to navigate away from the editor and there are unsaved changes, however, you should always ensure that your changes are saved before disconnecting the browser from the network or suspending a laptop.

Working with models

Overview of steps for adding a model

This topic gives a brief overview of how to add and design a new model. For more detailed information, see the topics that are referenced in the steps below.

You add and design a model as follows:

1. In the model manager, click **New model**. Enter a model name in the resulting dialog box. See also "Adding a new model" on page 34.
2. In the model editor, drag the required blocks from the palette onto the canvas. See also "Adding a block" on page 43.
3. Refer to the block documentation as necessary. See also "Viewing the documentation for a block" on page 46.
4. Use the block parameter editor to specify the parameters of the block. See also "Editing the parameters of a block" on page 44.
5. Connect the appropriate blocks with wires. See also "Adding a wire between two blocks" on page 48.
6. Save your changes. See also "Saving a model" on page 42.

   Note: Only saved models are listed in the model manager. When you add a new model and then leave the model editor without saving the model, it will not be listed in the model manager, and all the edits you made will be lost.

7. Leave the model editor. This takes you back to the model manager. See also "Leaving the model editor" on page 42.
8. A newly added model is automatically set to draft mode in the model manager. If you want to test it, simulate it, or make it available in production, see "Deploying a model" on page 35.

For detailed background information, including restrictions, see "Wires and Blocks" on page 61.

Changing the name, description, and tags of a model

You can rename each model that you are currently editing in the model editor, and you can also change the description of each model.

You can also add or remove tags. Tags are helpful in the model manager, to show only the models for which a specific tag has been defined (see also "Filtering the models" on page 33).

To change the name, description, and tags of a model

1. In the model editor, click on the model name which is shown at the left of the toolbar.
2. In the resulting Model Configuration dialog box, specify a new unique name for the model, change the description, and/or change the tags.

   To add a tag, you simply type its name and press Enter or the Tab key. The tag is then shown in a colored rectangle. To remove a tag, click on the that is shown in the rectangle. The dialog prevents you from entering duplicate tags for a model; if you enter such a tag name, the duplicate tag is not added and the original tag blinks one time.
3. Click **OK**.

Saving a model

When you save a model in the model editor, it is stored in the Cumulocity IoT inventory for your tenant, in JSON format.

Important: It may happen that you and another user are editing the same model at the same time. In this case, the changes that are saved last will be stored. So your changes might be overwritten by a later save by another user.

To save a model

- In the toolbar of the model editor, click .

  This toolbar button is only enabled when changes have been applied to the model and when the model has been given a name.

Leaving the model editor

When you leave the model editor using the corresponding toolbar button, you are returned to the model manager. You can then, for example, edit a different model, or change the mode or state of the current model.

Caution: All unsaved changes are lost when you navigate to a different URL or close the browser window.

To leave the model editor

- In the toolbar of the model editor, click .

In case there are still unsaved changes, you are asked whether to save or discard them.

Working with blocks and wires

Adding a block

The blocks in the palette are grouped into different categories. The blocks in the **Input** and **Output** categories represent devices, device groups and/or (if the search box is used) any assets that have been registered in the Cumulocity IoT inventory (visualized in the Device Management application). Other categories contain blocks, for example, for adding logic to the model or for adding calculations.

When you move the mouse pointer over a block in the palette, a tooltip appears which briefly explains the purpose of the block. The tooltip also shows the entire name of the block.

Horizontal scrollbars are available in both the **Input** or **Output** categories, below the tree. These scrollbars - and also the tooltip - are helpful when the name of a device, device group or asset is not fully shown in the palette.

Detailed information for each block is available in the block reference, which is shown in the documentation pane. See also "Viewing the documentation for a block" on page 46.

To add a block

1. In the palette of the model editor, expand the category which contains the block that you want to add.
2. When you expand the **Input** or **Output** category, the devices and device groups that are registered in the Cumulocity IoT inventory are initially shown.

   By default, up to 10 devices and 10 device groups are shown, sorted alphabetically. With a large inventory, you will have to click **Load more** to display any devices or device groups that are not shown initially.

   The palette reflects the parent/child hierarchy in the Cumulocity IoT inventory. The list of devices includes any defined child devices, and the list of device groups includes any defined sub-groups. These are available from expandable/collapsible nodes.

   The **Input** and **Output** categories both have a search box. These search boxes can be used to show any assets in the Cumulocity IoT inventory which match your search criteria. This includes devices, device groups and also managed objects. The search is case-sensitive. You can enter a name or part of a name. The characters that you type in may be contained at any position within the name. For example, enter the word "test" to show only the assets that have this word in their names. You need not press Enter; the list is updated with each character that you type. The search result is sorted alphabetically. With a large search result, you will have to click **Load more** to display any assets that are initially not shown.

The maximum number of shown devices, device groups and/or assets depends on a tenant option. For more information, see "Configuring the number of shown devices, device groups and/or assets" on page 84.

The assets that are shown when searching also depend on a tenant option. You can restrict the search to show only assets of a specific type. For more information, see "Searching for input and output assets" on page 85.

When you expand a different category, your search criteria are remembered.. This is helpful if you want to have shorter lists for the **Input** and **Output** categories, both with individual contents as defined by your search criteria.

The **Output** category offers for selection a special block, the **Trigger Device**. This sends the output back to the device which triggered the output.

3. Drag the block from the palette and drop it on the canvas.

   When you drop the block on an existing block on the canvas, the new block is created on top of that block. When you drop the block on a collapsed group, the new block is created below that group. In both cases, you should move the new block to a free space of the canvas. See also "Moving a block" on page 47.

   When you drop the block on an expanded group (where the contents of the group are visible), the new block is added to that group. For more information on groups, see "Working with groups" on page 52.

   Note: Even though the **Output** category shows device groups, it is not possible to drag them onto the canvas because group output is not allowed.

4. Specify all required parameters for the block. See "Editing the parameters of a block" on page 44.

   Note: The block parameter editor is automatically shown when you add a block for which parameters need to be specified. It is not shown, however, if the block does require any parameters (such as the **OR** block).

Editing the parameters of a block

Most blocks (but not all) have parameters that you have to set according to your requirements.

When "Missing" is shown on an input or output block on the canvas, this means that the defined device, device group or asset cannot be found in the Cumulocity IoT inventory. You should then either go to the Cumulocity IoT inventory and make sure that this device is registered or that the device group or asset exists, or you should select a different, existing device, device group or asset in the block parameter editor (see below).

The labels of some blocks on the canvas show the value of the most important parameter. For example, the **Expression** block shows the defined expression, and the **Time Delay** block shows the defined delay in seconds.

The block parameter editor also contains commands for duplicating and deleting the currently selected block. See "Duplicating a block" on page 48 and "Deleting a block or wire" on page 49 for detailed information.

For the input and output blocks, you can globally replace the devices, device groups or assets that are used. See "Replacing devices, device groups and assets" on page 50 for detailed information.

To edit the parameters of a block

1. On the canvas of the model editor, click the block that you want to edit using the *left* mouse button.

   The block parameter editor appears, providing input fields for all parameters that can be specified for that block.

2. For some blocks (such as an input block or output block for a device), the block parameter editor shows a **Block Type** drop-down list box. Select the type of block that is appropriate for your requirements.

   Note: The following applies when you change the block type for a block that is already wired to one or more other blocks: if the new block type has different port names (for example, if the port name changes from **Value** to **Value 1**), the existing wires to/from the changed ports are removed. This is done because a changed port name would make the existing wiring invalid.

3. For the input and output blocks, you can select a different device (for input or output), device group (for input only) or an asset (if the search box is used) from a dialog box.

   By default, 10 devices and 10 device groups are shown, sorted alphabetically. With a large inventory, you will have to click **Load more** to display any devices or device groups that are not shown initially.

   The tree in the dialog box reflects the parent/child hierarchy in the Cumulocity IoT inventory. The list of devices includes any defined child devices, and the list of device groups includes any defined sub-groups. These are available from expandable/collapsible nodes. For output blocks, you can also select a **Trigger Device**.

   The search box can be used to show any assets in the Cumulocity IoT inventory which match your search criteria. This includes devices, device groups and also managed objects. The search is case-sensitive. The characters that you type in may be contained at any position within the name. The tree is updated with each character that you type. With a large search result, you will have to click **Load more** to display any assets that are initially not shown.

Click the **Use** button (this is shown when you move the mouse over an entry) to select the device, device group or asset that you want to use. For output blocks, you cannot select a device group, but you can select any device within that group.

The maximum number of shown devices, device groups and/or assets depends on a tenant option. For more information, see "Configuring the number of shown devices, device groups and/or assets" on page 84.

The assets that are shown when searching also depend on a tenant option. You can restrict the search to show only assets of a specific type. For more information, see "Searching for input and output assets" on page 85.

4. For some blocks (such as the **Range Lookup** block), the block parameter editor shows text boxes for specifying key-value pairs. If you need to specify more key-value pairs, click **Add row**. The key-value pair in the first row is processed first. You can drag a row to a different position using the ⇅ control that is shown next to that row, and you can delete a row that you do not need any more by clicking ✕ next to that row. Empty rows are automatically removed when you leave the block parameter editor.

5. Specify all required parameters.

   Detailed reference information for each block (and block type) is available from the documentation pane. See also "Viewing the documentation for a block" on page 46.

   Your input is kept in memory when you leave the block parameter editor (for example, when you click on another block or the canvas).

   Note: Keep in mind that your changes are only written to the inventory when you save the model. See also "Saving a model" on page 42.

Viewing the documentation for a block

The documentation pane allows you to view detailed information for the currently selected block. It shows the so-called *Block Reference* which provides documentation of a block's parameters, input ports and output ports. You can resize the documentation pane, and you can also toggle its display.

To view the documentation for a block

1. In the model editor, click the block for which you want to view the documentation. You can do this in the palette or on the canvas.
2. When a **Block Type** drop-down list box is shown in the block parameter editor, select the block type for which you want to view the documentation.
3. If the documentation pane is currently not shown, click the area that contains the icon (shown at the right of the canvas) to display the documentation pane. Clicking that area again hides the documentation pane.

4. If you want to resize the documentation pane (for example, to make it larger), move the mouse pointer over the area that contains the icon. When the mouse pointer changes to show the resize icon (←‖→), click and hold down the mouse button and drag the mouse to the left or right (to make the documentation pane wider or smaller).

Selecting blocks and wires

If you want to move, duplicate or delete one or more blocks that are currently shown on the canvas of the model editor, you first have to select the required blocks.

To select a single block on the canvas, you just need to click the block. With a block, the resulting behavior depends on the mouse button that you use:

- When you click the block using the *left* mouse button, the block is selected and the block parameter editor is shown (see also "Editing the parameters of a block" on page 44).
- When you click the block using the *right* mouse button, the block is selected only (the block parameter editor is not shown). This is helpful if the editor would be in the way, for example, when adding a wire to another block.

To select a single wire, you just need to click the wire (you can use either mouse button in this case).

To select several blocks and/or wires at the same time, do one of the following:

- Press Ctrl and click each block and/or wire that you want to select.
- Or to select an area containing several blocks and wires, click and hold down the mouse button over an empty space of the canvas and wait until the mouse pointer changes to a cross. Then drag the mouse to select the desired area. Release the mouse button when all required blocks and wires have been selected.
- Or to select all blocks and wires, press Ctrl+A.

To deselect your selection:

- Press Ctrl and click the currently selected block or wire.
- Or to deselect all selections, click an empty space of the canvas.

Moving a block

You can move each block that is currently shown on the canvas to different location. When one or more wires are attached to a block that is moved, the wires are also moved.

To move a block

- On the canvas of the model editor, click the block that you want to move, hold down the mouse button and drag the block to the new location.

- Or to move several blocks at the same time, select them as described in "Selecting blocks and wires" on page 47. Then click and hold down the mouse button and immediately drag the blocks to the new location (do not wait until the mouse pointer changes).

Duplicating a block

You can duplicate each block that is currently shown on the canvas. The original block and its copy will then both have the same parameters.

When you duplicate a single block, the attached wires are not automatically duplicated. When you duplicate several blocks at the same time, however, the attached wires between the selected blocks are automatically duplicated.

To duplicate a block

- On the canvas of the model editor, click the block that you want to duplicate and then do one of the following:
  - Click the **Duplicate** command which is shown at the bottom of the block parameter editor.
  - Or press Ctrl+C to copy the block, and then press Ctrl+V to paste the block.
  - Or press Ctrl and drag the block to be duplicated to the position at which you want to place the copy.
- Or to duplicate several blocks at the same time, select them as described in "Selecting blocks and wires" on page 47 and then proceed as described above. Exception: the **Duplicate** command is only available when you select a single block.

Adding a wire between two blocks

The blocks on the canvas can be wired together to indicate that the output from one block is used as the input for the other block.

The wires are attached to *ports*, that is, to the circles that are shown to the left and/or right of a block. Each block can have zero, one or more of the following:

- output ports (shown at the right side of a block)
- input ports (shown at the left side of a block)

To see the labels of the ports, click the block to select it. Or move the mouse pointer over a port to see the label in a tooltip.

See "Wires and Blocks" on page 61 for detailed information on the types of values that can be sent between two blocks, the processing order of wires, restrictions, and more.

To add a wire between two blocks

- On the canvas of the model editor, click the output port of the block that you want to connect and drag the mouse to the input port of another block.

Changing a wire

You can change the path that a wire takes to the block to which it is currently connected. And you can also rewire a block so that it is connected to a different block or to a different port of the same block.

Wires cannot create cycles. See "Wire restrictions" on page 71 for detailed information.

To change a wire

1. On the canvas of the model editor, click the wire that you want to change.

   The port names of the attached blocks are then shown, and the ports attached to each end of the wire are highlighted.

2. To change the path that a wire takes between two blocks, drag one of the resize icons (■) that are now shown on the selected wire to a different position.

   Or to move the wire to a different port, drag the move icon (◆) that is now shown at the input or output port (a hand pointer is shown in this case) to a different port.

Deleting a block or wire

You can delete each block or wire that is currently shown on the canvas. When you delete a block, all wires that are attached to this block are automatically deleted.

To delete a block or wire

- On the canvas of the model editor, click the block or wire that you want to delete and press Del.

  In the case of a block, you can alternatively click the **Delete** command which is shown at the bottom of the block parameter editor.

- Or to delete several blocks and/or wires at the same time, select them as described in "Selecting blocks and wires" on page 47 and then press Del.

Undoing and redoing an operation

You can undo and redo each change that has been applied to the canvas. For example, you can undo the deletion of blocks, undo changed parameter values, or undo the rerouting of a wire.

It is not possible to undo/redo the change to a model name or its description.

Note: To use the key combinations mentioned below, the canvas must have the focus. When the documentation pane or the palette currently has the focus, the change on the canvas is not undone/redone.

To undo or redo an operation

- To undo the last operation, click ↶ in the toolbar of the model editor.

  Or press Ctrl+Z.
- To redo the last operation, click ↷ in the toolbar of the model editor.

  Or press Ctrl+Y.

The above toolbar buttons are only enabled when there is an operation that can be undone or redone.

Replacing devices, device groups and assets

You can search the input and output blocks for the devices, device groups and assets that are used in the current model and replace them with other devices, device groups or (if the search box is used) assets that are currently registered in the Cumulocity IoT inventory (visualized in the Device Management application).

Note: In the rules below, the term *device* refers to a device or other asset (but not to a device group).

The following rules apply:

- You can replace a device with another device.
- You can replace a device group with another device group.
- When you replace a device with a device group:
  - all matching input devices are changed to device groups, and
  - all matching output devices are changed to trigger devices.
- When you replace a device group with a device:
  - a device group is changed to a device, and
  - all matching trigger devices are changed to the specified device.

    Note: If you change more than one device group to a device at a time, then only the first specified device will be used to replace all trigger devices.
- An entry named "Trigger Device" is not available for selection in the dialog.

After you have replaced the devices, you need to verify that the measurements that are used by the input and output blocks of the current model still refer to the appropriate measurements. The Cumulocity IoT fragment and series are not changed by the replacement, which may or may not apply to the newly defined device.

To replace devices, device groups and assets

1. In the toolbar of the model editor, click . This toolbar button is only enabled when at least one device, device group or asset has been defined in the current model. Any defined trigger devices are not considered in this case.
2. In the **Current device and device group** drop-down list box of the resulting dialog box, select the device, device group or asset that you want to replace. All devices, device groups and assets that are used in the model are available for selection.
3. Click the **Replace with** box to display a dialog box. The dialog box is the same as when selecting a different device, device group or asset in the block parameter editor. See "Editing the parameters of a block" on page 44 for more information on this dialog box. Click the **Use** button (this is shown when you move the mouse over an entry) to select the device, device group or (if the search box is used) asset that you want to use instead.

   The maximum number of shown devices, device groups and/or assets depends on a tenant option. For more information, see "Configuring the number of shown devices, device groups and/or assets" on page 84.

   The assets that are shown when searching also depend on a tenant option. You can restrict the search to show only assets of a specific type. For more information, see "Searching for input and output assets" on page 85.
4. If you want to replace further devices, device groups or assets, click **Add row**. This is only shown if more than one device, device group or asset has been defined in the current model.

   A new row is shown, containing additional **Current device and device group** and **Replace with** drop-down list boxes, and you can now select one more device, device group or asset to be replaced. Any devices, device groups and asset that you have previously selected for replacement are no longer offered for selection in the **Current device and device group** drop-down list box.

   Repeat this step until all required devices, device groups and assets have been selected for replacement. You can add as many rows as there are devices, device groups or assets in the current model.
5. If you want to remove a row (for example, when you no longer want to replace a selected device), click ✕ next to that row. This is only available if the dialog box currently shows more than one row.
6. Click **Replace**.

Copying items to a different model

You can copy any items on the canvas (blocks, groups, and attached wires) and paste them in a different model. The prerequisite for this is that all is done in the same session. It will not work if you try to paste the items in a different tab or in a different browser.

Caution: There may be performance issues if you copy many input blocks and output blocks. This is because this operation requires access to the inventory service of Cumulocity IoT to get the information about the devices that are represented by these blocks.

To copy items to a different model

1. On the canvas of the model editor, select all items that you want to copy and press Ctrl+C.

   This also works if the model is currently in read-only mode.

2. Leave the model editor. See also "Leaving the model editor" on page 42.
3. In the model manager, switch to the model into which you want to paste the copied items. This can be an existing model (see also "Editing an existing model" on page 35) or a new model that you first have to create (see also "Adding a new model" on page 34).
4. When the model editor is shown, press Ctrl+V to paste the copied items into the model.

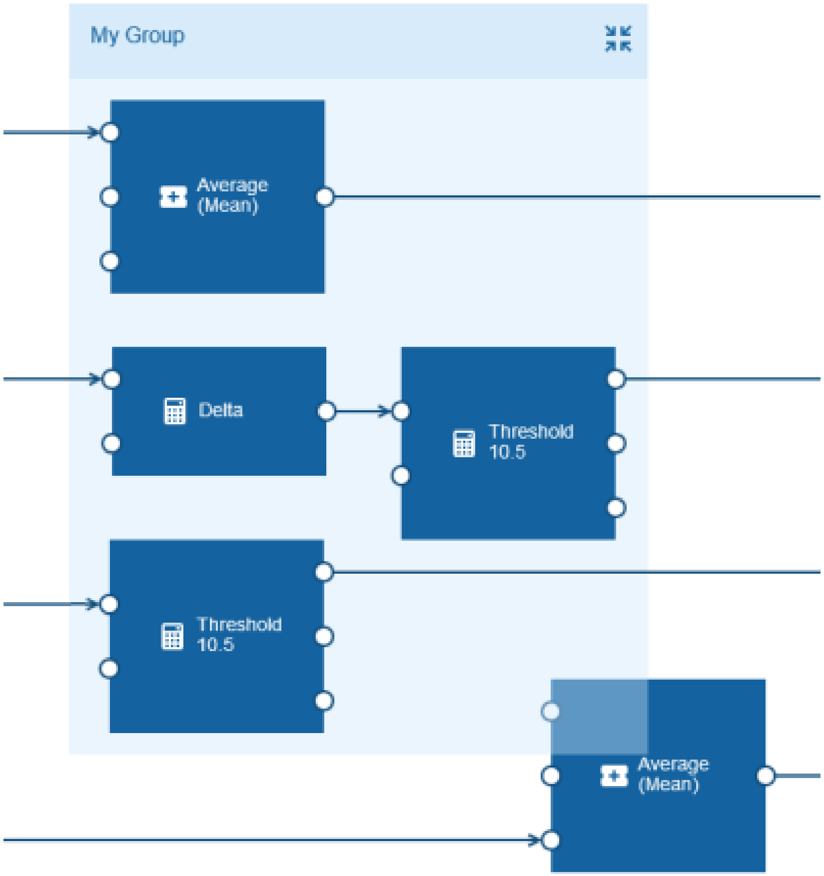
Working with groups

What is a group?

You can arrange blocks and their attached wires in a group. A group is a special type of block which can be collapsed and expanded. When a group is expanded, you can change its contents in the same way as you would on the canvas, for example, you can add wires or edit the block parameters. You can also add more blocks to the group or remove blocks from the group. When a group is collapsed, it occupies less space on the canvas, however, the blocks and wiring within the group are not visible in this case.

Groups are helpful if commonly required functionality needs to be made available in multiple places. You can give each group a name by which it can be identified. You can copy a group and paste it in either the same model or in a different model.

Note: Do not confuse this type of group with a device group. A device group is a special input block that is offered for selection from the palette. See also "Adding a block" on page 43.



The size of the box that is shown for a group is determined by its contents. If you move a block within the group to a different position, the box size is automatically adapted (that is, the box is made larger or smaller). The same applies if you change the path that a wire takes to another block within the same group.

You move a group on the canvas in the same way as you move a block (see also "Moving a block" on page 47). When you move a group, the group is always shown on top of all other items on the canvas. As the group box is transparent, you can easily see which blocks belong to the group and which are just overlayed by the box.

It is not possible to nest groups.

Note: There is one exception when managing the contents of a group: When you copy one or more blocks that are contained in a group using Ctrl+C and Ctrl +V or if you use the **Duplicate** command in the block parameter editor, the copy is not added to the group. It is added to the canvas instead. However, when you press Ctrl and then drag the blocks to be duplicated, you can place the copy either within the group (this can be the same or a different group) or on the canvas. See also "Duplicating a block" on page 48.

Adding a group

You can add any blocks that are currently shown on the canvas (including the wires between the blocks) to a group.

It is not possible to create an empty group. You first have to add a group as described below. Once the group exists, you can add more blocks to the group, either from the palette or from the canvas, as described in "Adding a block" on page 43 and "Moving blocks into a group" on page 55.

To add a group

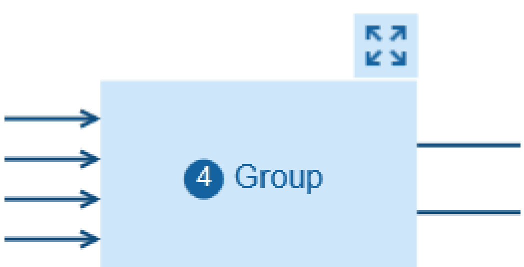
1. On the canvas of the model editor, select one or more blocks that you want to add to a group. You need not select wires; all existing wires are retained. See also "Selecting blocks and wires" on page 47.
2. In the toolbar of the model editor, click .

   Or press Ctrl+G.

Collapsing and expanding a group

If you need more space on the canvas and do not need the group contents to be visible, you can collapse the group.

When a group is collapsed, a number is shown on the collapsed group indicating the number of blocks in that group. For example:



If you want to make the group contents visible again (for example, to edit block parameters or to add wires), you have to expand the group.

When you save the model, the state of each group (that is, whether it is currently collapsed or expanded) is stored. The next time you edit the model, its contents will be shown as after the last save.

To collapse or expand a group

- To collapse a group, click which is shown next to the group name.
- To expand a group, click which is shown above the top right of the collapsed group.

Renaming a group

When you add a group, its default name is "Group". You can rename each group and give it a unique name.

If a group name is longer than can be shown in the group label, move the mouse pointer over the group name to view the entire name in a tooltip.

It is not possible to have groups without names. If you delete a group name, the previous name is automatically used again.

To rename a group

1. In the model editor, select the group and then click on the group name. You can either do this when the group is collapsed or expanded (see also “Collapsing and expanding a group” on page 54). This selects the entire name for editing.
2. Specify a new group name and press Enter.

Moving blocks into a group

You can move one or more blocks from the canvas into an existing group. All existing wires are retained.

You can also drag a block from the palette into an existing group. See “Adding a block” on page 43.

You can only move/drag blocks into a group when its contents are visible, that is, when the group is currently expanded. See also “Collapsing and expanding a group” on page 54.

To move blocks into a group

1. Make sure that the group into which you want to move the blocks is not collapsed.
2. On the canvas of the model editor, select the blocks that you want to move into the group (see also “Selecting blocks and wires” on page 47). You need not select the wires between the blocks; they are automatically moved together with the blocks.
3. Do one of the following:
   - Drag the selection into the group and drop it there.
   - Or select the group into which you want to move the blocks. Then click  in the toolbar of the model editor, or press Ctrl+G.

Removing blocks from a group

When you remove a block from a group, the block is moved to the canvas. It is not deleted. All existing wires are retained.

When the last item of a group has been removed, the group is automatically deleted. If you want to remove all items from a group at the same time, you can simply ungroup the entire group. See “Ungrouping a group” on page 57.

To remove blocks from a group

- To remove one or more blocks from a group at the same time:
  1. In the expanded group, select the blocks that you want to remove.
  2. In the toolbar of the model editor, click . Or press Ctrl+Shift+G.
- Or to remove a single block from a group:
  1. In the expanded group, select the block that you want to remove.
  2. Click the **Remove from Group** command which is then shown at the bottom of the block parameter editor.

Deleting blocks and wires from a group

You delete blocks and wires from a group in the same way as deleting them directly on the canvas. The only prerequisite is that the group is currently expanded. See “Deleting a block or wire” on page 49.

If the last item in a group is deleted, the group is automatically deleted.

Copying a group

You can copy each group that is currently shown on the canvas. The original group and its copy will then both have the same contents.

Wires coming in from blocks outside of the group or going from the group to blocks outside of the group are not copied.

You can also copy a group into a different model, see “Copying items to a different model” on page 51.

To copy a group

- On the canvas of the model editor, click the group that you want to copy (it does not matter whether the group is currently collapsed or expanded) and then do one of the following:
  - Press Ctrl+C to copy the group, and then press Ctrl+V to paste the group.
  - Or press Ctrl and drag the group to be copied to the position at which you want to place the copy.

Ungrouping a group

When you ungroup a group, the group is removed and all the blocks from that group are shown directly on the canvas. All attached wires are retained.

You can ungroup several groups at the same time. In this case, it is important that no block or wire is selected either within or without the selected groups, otherwise ungrouping is not possible.

To ungroup a group

1. On the canvas of the model editor, select one or more groups that you want to ungroup. It does not matter whether a group is currently collapsed or expanded.
2. In the toolbar of the model editor, click .

   Or press Ctrl+Shift+G.

Deleting a group

You can delete each group that is currently shown on the canvas.

Caution: When you delete a group, all blocks and wires within this group are also deleted.

To delete a group

- On the canvas of the model editor, click the group that you want to delete (it does not matter whether it is currently collapsed or expanded) and press Del.

Managing the canvas

Navigating large models

If your model is too large to fit onto the visible part of the canvas, you can use the mouse to drag the parts of the model into view that are currently outside of the window. You can do this either directly on the canvas or in the overview area. The overview area always shows the entire model. If the overview area is currently not shown, see "Showing and hiding the overview" on page 58.

To navigate in a large model

1. In the model editor, position the mouse over a free spot of the canvas (which does not contain a block or wire) or anywhere over the overview area.

2. Click and hold down the mouse button, and immediately drag the mouse into the desired direction. Release the mouse button when the required area is visible on the canvas.

Note: When you hold down the mouse button for a longer time over a free spot of the canvas, the mouse pointer changes and you can select an area instead (for example, several blocks and attached wires). See also "Selecting blocks and wires" on page 47.

Showing and hiding the overview

The overview area, which shows the entire model, is shown at bottom right of the canvas. If you do not need the overview, you can hide it.

To show or hide the overview

- To hide the overview, click which is shown directly above the overview area.
- To show the overview, click at the bottom right of the canvas.

Zooming the canvas

The toolbar of the model editor indicates the current zoom percentage for the canvas. The zoom buttons in the toolbar allow you to

- zoom out, which makes everything on the canvas smaller so that more items can be shown, and to
- zoom in, which makes everything on the canvas larger, but less items can then be shown.

Note: When you use the key combinations mentioned below, the currently selected area defines what is to be zoomed. When the canvas has the focus (for example, when you have just selected a block or wire), only the content of the canvas is zoomed. When the documentation pane or the palette currently has the focus, the browser's zoom functionality is used and all of the browser content is zoomed (and the zoom percentage in the toolbar remains unchanged).

To zoom the canvas

- To zoom out, click in the toolbar of the model editor.

  Or press Ctrl and the minus key.
- To zoom in, click in the toolbar of the model editor.

  Or press Ctrl and the plus key.

Showing and hiding the grid

The blocks, wires and groups on the canvas always snap to a grid. You can decide whether the grid is to be shown or not. The grid is not shown by default. When you zoom the canvas, the grid is zoomed accordingly.

To show or hide the grid

- Click  in the toolbar of the model editor to toggle the display of the grid.

  The toolbar button always indicates whether the grid is currently hidden or shown. When the grid is hidden, the button looks as shown above. When the grid is shown, the button looks as follows: .

  If the model is in the Active state (read-only mode), it is not possible to toggle the display of the grid and this button is therefore disabled.

6 Wires and Blocks

Values sent on a wire

Blocks within a model are connected from block outputs to block inputs with wires.

Note: These block outputs and inputs are also called *output ports* and *input ports*. See also "Adding a wire between two blocks" on page 48.

Wires allow blocks to pass signals and values between blocks. The value sent on a wire is one of the following types, according to the block output from which it is connected:

Type	Description
`boolean`	A true or false value. A Boolean value stays true or false until changed.
`float`	A numeric value, which can be fractional (and processed using fixed precision). A float value maintains its current value until changed.
`string`	A textual value. A string value maintains its current value until changed.
`pulse`	A signal of a point in time. Pulses are only active momentarily. Unlike the above types, they only represent a single instance in time. See also "The pulse type" on page 68.
`any`	A value that may be any of the above types. See also "The any type" on page 68.

The type of a wire depends on the output to which it is connected. This can be viewed in the block reference. Similarly, the type (or supported types) of a block's input can be viewed in the block reference.

Value types

The following types are referred to as *value types*:

- `boolean`
- `string`
- `float`
- `any` when used to hold a `boolean`, `string` or `float` value

Value types are useful for modeling measurements such as sensor values, which may be read intermittently, or sampled. In between readings, the physical property being measured (such as temperature) will still have some value, as it is a continuous property. For practical reasons, a sensor may not give a continuous stream of output but instead a periodic sampling, or provide new readings only if the value being measured has changed (within whatever measurement resolution the sensor provides). Between sample points, blocks will use the most recent value, as that is the most up to date value being provided. In general, blocks assume that a value stays at whatever the most recent reading of that value is until a new value is received.

For example, consider a pair of temperature sensors. One provides a reading every 10 seconds regardless, while another only provides a new reading if the value has changed by 0.5 degrees. If we connect these to a **Difference** block, then we may have inputs as shown in the following table, with the corresponding result from the **Difference** block's **Absolute Difference** output:

Time	Sensor 1 (reads every 10s)	Sensor 2 (output if changed by 0.5)	Difference block: Absolute Difference output
10:00:00	20.0		
10:00:03		22.0	2
10:00:10	20.0		2
10:00:20	20.0		2
10:00:23		22.5	2.5
10:00:28		23.0	3
10:00:30	21.1		1.9
10:00:35		23.5	2.4
10:00:40	22.8	24.0	1.2

Note that two inputs (to different input ports of the block) to the same block with the same timestamp only generate a single output. For each wire within a model (and each input block), there can only be a single value for a given point in time. An input block cannot generate more than one output for the same timestamp. If it receives multiple events at the same time, then it is undefined which of the events is picked.

In general, blocks will not consider there to be any significance to a wire receiving the same `boolean`, `float` or `string` value as before. Most blocks will not change behavior.

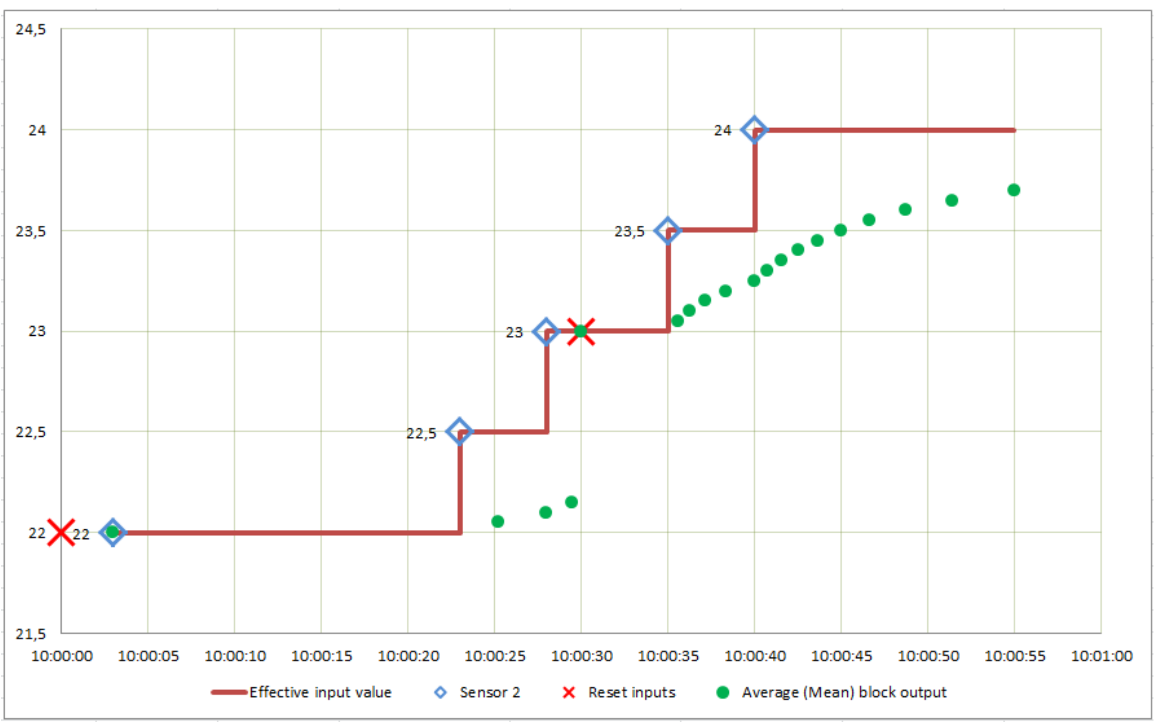
This is true for any arithmetic blocks, such as the **Difference** block in the example above: the output is still 2 on the repeated readings from sensor 1. There are some exceptions, such as the **Missing Data** block when the **Ignore Repeated Inputs** check box is not selected (`false`).

If a single block has a numeric value input and pulse signals such as reset, the absence of a new value when a pulse signal occurs means that the value is treated as having the same value still. Thus, when an **Average (Mean)** block is reset, its output will be equal to the most recently received input (assuming it has received an input since the model has started). In the example below, the **Average (Mean)** block's duration has not been set, while the output threshold is set to 0.05; this means the block will generate new output even if there is no new input (see "Common block inputs and parameters" on page 72).

Time	Reset signal	Sensor 2	Average (Mean) block output	Notes
10:00:00	Reset			No output. There has been no input value yet.
10:00:03		22.0	22.00	With no history, the output value is the input value.
10:00:23		22.5		All of the values up to this point have been 22, so the average value is still 22 (thus, no new output is generated).
10:00:25.22			22.05	Average of 20 seconds at value 22 and 2.22 seconds at value 22.5.
10:00:28		23.0	22.10	Average of 20 seconds at value 22 and 5 seconds at value 22.5.
10:00:30	Reset		23.00	The input is still 23 (we just have not received a new event), and reset only discards the history. With no history, the output value is the input value.
10:00:35		23.5		

Time	Reset signal	Sensor 2	Average (Mean) block output	Notes
10:00:35.56			23.05	Average at various points in time, when the output changes by 0.05.
10:00:36.25			23.10	
10:00:37.14			23.15	
10:00:38.33			23.20	
10:00:40		24.0	23.25	Average of 5 seconds at value 23 (from reset at :30 to :35) and 5 seconds at value 23.5 (from :35 to :40).

The following graph illustrates the inputs to the **Average (Mean)** block and the output of this block:



Note how the effective input value is unchanged until a new measurement input occurs, and the **Average (Mean)** block operates on this effective value (the red line in the above graph). When reset, the block outputs the current effective input, which at the second reset at 10:00:30 is 23. Note that when the **Output Threshold** parameter is set, new outputs can be generated even if no new input occurs, and will asymptotically approach the last input value. Note that this behavior differs from Apama queries or stream queries.

If the **Average (Mean)** block was configured with a window of 10 seconds, then the window would apply as illustrated below:

Time	Reset signal	Sensor 2	Effective input value	Average (Mean) block output	Values in window history	Notes
10:00:00	Reset					
10:00:03		22	22	22.00		First value after start: the window is empty, so the **Average (Mean)** block uses the input value for the output.
10:00:23		22.5	22.5		22	
10:00:23 - 10:00:28			22.5	increasing from 22.00 to 22.20	22, 22.5	Proportion of window that is 22 or 22.5 changes over time, thus the output changes.
10:00:28		23	23	22.25	22, 22.5	
10:00:28 - 10:00:30			23	increasing from 22.25 to 22.40	22, 22.5, 23	
10:00:30	Reset		23	23.00		Window is reset and thus now empty; the current (effective) input is 23, so the **Average (Mean)** block uses that for the output.
10:00:35		23.5	23.5		23	

Time	Reset signal	Sensor 2	Effective input value	Average (Mean) block output	Values in window history	Notes
10:00:35 - 10:00:40			23.5	increasing from 23.00 to 23.20	23, 23.5	
10:00:40		24	24	23.25	23, 23.5	Window is now full (10 seconds since reset).
10:00:40 - 10:00:45			24	increasing from 23.25 to 23.75	23, 23.5, 24	
10:00:45			24	23.75	23.5, 24	Value 23 is now finally expired from the window (this was the effective input until 10:00:35, which is 10 seconds ago).
10:00:45 - 10:00:50			24	increasing from 23.75 to 24	23.5, 24	
10:00:50			24	24	24	Value 23.5 is now finally expired from the window (this was the effective input until 10:00:40, which is 10 seconds ago). The window now contains 10 seconds worth of measurements, all with value 24.

In the above, note how the current value only has any weighting in the window (that is, contributing to the output value) after the measurement is received. At the point the measurement is received, it has zero weighting compared to the previous history. As before, the sensor's value remains the effective input until it is replaced with a newer value (note that this is different to aggregates with timed-based windows in Apama queries or stream queries). For example, the block has an effective input value of 23.5 from 10:00:35 to 10:00:40, and the value 23.5 is thus only finally expired from the window at 10:00:50, 10 seconds after it ceased to be the current effective input value, rather than 10 seconds after it first entered the window. Finally, note that when the window is empty, the effective input is used as the output instead, as the window is zero-length.

The pulse type

In contrast to value types, the `pulse` type represents a single point in time. For example, this may be a result of:

- a user pressing a momentary-action button,
- a state transition of a device,
- a sensor detecting a person walking through a door,
- a heartbeat event to denote a remote device is still alive, or
- a state transition of a block within a model.

Typically, blocks act upon every pulse sent to one of their inputs. Pulses are commonly used to trigger an output from a model using an output block, or used to reset the state of blocks within a model.

Pulses are active momentarily. In some regards, they are similar to a Boolean value which is automatically reset to `false` after a model has processed a value.

Repeated pulses are typically significant, though they may not necessarily result in any change, depending on how they are being used. For example, repeatedly resetting an **Average (Mean)** block while its input value is unchanged will result in the output value remaining the same.

The any type

The `any` type is used on blocks which pass through a value of any type (for example, a **Time Delay** block or a **Gate** block).

Values of the `any` type can represent a value type or a `pulse` type.

Type conversions

It is legal to connect a block output to a block input if they are the same type. Most other connections are also permissible, which result in the conversions as described in the table below. An ✘ indicates that a connection is not legal; trying to deploy a model with such a wiring connection will fail.

		From block with output type				
		pulse	**boolean**	**float**	**string**	**any**
Connect to input of type	**pulse**	✔	pulse occurs when output changes to true	pulse occurs when output changes value	pulse occurs when output changes value	pulse occurs when output changes value (excluding changes to false)
	boolean	true when the pulse has occurred, otherwise false	✔	true if non-zero	true if not an empty string	true if value non-zero/ empty
	float	✘	0 for false, 1 for true	✔	✘	✘
	string	✘	"true" or "false"	number converted to a string (may be in scientific notation)	✔	string value (may be in scientific notation)
	any	✔	✔	✔	✔	✔

Only conversions that will always succeed are allowed. String values are not converted to float values; while the input conversion may work sometimes, it cannot be guaranteed to always work.

In many cases, you need not worry about type conversions and where a wire makes sense. Any type conversion that is needed happens automatically.

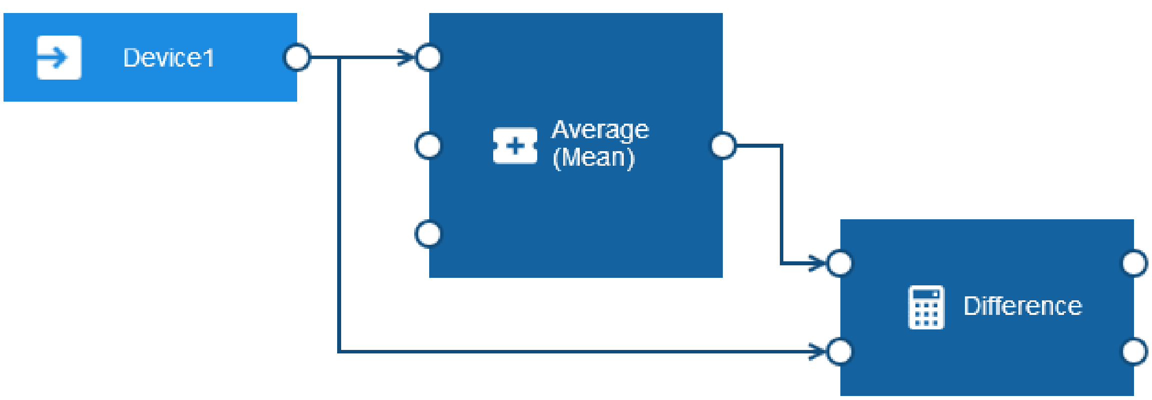
Some blocks accept different types of inputs, and may change their output type or behavior depending on the input types. For example, the logical **OR** block can operate on either Boolean or pulse inputs, and its output is the same as its input types.

In some cases, it is desirable to force a value to be interpreted as a specific type, in which case a converter block can be used to force a conversion to a specific type. For example, the **Pulse** block can convert Boolean or float values to pulses, according to the conversions above. This means: for Boolean, generate a pulse when the Boolean value changes to true; for float, generate a pulse when the value changes. Thus, connecting two float outputs to an **OR** block directly will generate a Boolean output which is true when either of the float outputs is non-zero. Alternatively, connecting two float outputs each to a **Pulse** block and from them to the inputs of an **OR** block, will send a pulse whenever either float output changes value.

Processing order of wires

Where a block has multiple inputs connected, all of these inputs are calculated before the block performs any calculations based on the inputs. It may be that the inputs for a block occur out of step with each other (such as in the example for two temperature sensors in "Value types" on page 62), in which case a block uses the latest value for value type inputs.

Where a single value is sent on two or more paths which both lead to the same block, the block performs calculations based on the latest value for both paths. This ensures consistent behavior when multiple paths to a single block exist. For example:



When the device measurement is received, the **Average (Mean)** block calculation is completed to generate an average before the **Difference** block computes the difference between the value and its average.

Wire restrictions

While a block's output can be connected to multiple other blocks, a block's input can only have a single connection.

It is also legal to leave a block's input or output unconnected if that is not required (the **Average (Mean)** block in the example that is given in "Processing order of wires" on page 70 does not have anything connected to its **Sample** or **Reset** inputs).

Wires cannot create cycles. This means, the output of a block cannot be connected to

- the input of the same block, or to
- the input of any block that is connected directly or indirectly to one of the source block's inputs.

For example, there are three blocks: Block1, Block2 and Block3. A model would contain a cycle in the following cases:

- The output of Block1 is connected to the input of Block2, and the output of Block2 is connected to the input of Block1.
- The output of Block1 is connected to the input of Block2, the output of Block2 is connected to the input of Block3, and the output of Block3 is connected back to the input of Block1.

There are many possible connections which may lead to cycles in the model. The model editor, however, prevents you from creating cycles.

Block inputs and outputs

Many blocks have inputs or outputs that do not have to be used.

Some blocks generate several different outputs, and a model may only require some of the outputs available.

Some blocks have inputs, especially inputs of the `pulse` type, which do not have to be used. Leaving these not connected to anything is fine, and the operation associated with those inputs (such as **Reset**, see "Common block inputs and parameters" on page 72) will never be triggered.

Blocks can, when needed, detect which inputs are connected. For example, the **AND** block has five inputs, but it only requires the inputs that are connected to be `true` to generate a `true` output.

Common block inputs and parameters

The inputs listed below are the names of common input ports that are shown on the left side of a block.

- **Value** input

  Most calculation blocks have one main input which is called **Value**. This is the value on which the block performs its main calculation.

- **Value 1** and **Value 2** inputs

  Blocks may have a number of similar inputs, which may be labeled **Value 1**, **Value 2**, and so on. You can find such inputs with the **Difference** block (see also the example in "Value types" on page 62) or with the **AND** and **OR** logic blocks. Typically, there is nothing significant as to which input is used.

- **Reset** input

  Blocks that maintain some internal state may also have a **Reset** input, which is typically a `pulse` type. This does not have to be connected, but can be used to explicitly control on which range of readings a block should perform a calculation. For example, a model that monitors vehicle journeys may reset on the engine starting, which signifies the beginning of a journey. See also "Value types" on page 62 for an example that illustrates the **Reset** input.

- **Sample** input and **Output Threshold** parameter

  Blocks typically re-calculate their output when a new input is received. Some blocks may also generate output at some point after receiving an input, either because of time delay parameters set (for example, with the **Missing Data** or **Time Delay** blocks), or because their output may change over time even if the input value is constant. For example, the **Integral** block with a positive input generates an ever-increasing output until its window is full (or indefinitely if no duration has been set, when the block is calculating the integral over an unbounded window).

  As with real-world sensors, it is not practical to create a continuously changing output. As well as generating an output if their input value changes, such blocks may also have a **Sample** input which triggers the block to re-evaluate and generate a new output, even if the input has not received any new value and the output has not changed by a significant amount. This is useful if there is a specific point in time when the output of the block should be calculated, as its output is going to be used at a later point in the model.

  Alternatively, such blocks may have an **Output Threshold** parameter, which is used to control how frequently the output is re-calculated. When set, the block determines when its output will change by the output threshold, and when that occurs, even if it is not as a result of any new input value, the block generates an output value.

  The **Output Threshold** should be set taking into account what error margins will exist on the input value (real-world physical sensors have some limited precision and accuracy in the property they are measuring), and what precision is required in the output.

Take care to avoid **Output Threshold** values that are too large or too small. If the values are too large, the block does not generate a new output when needed (unless the **Sample** input is used). If the values are too small, the block limits how frequently it generates output. If you want to change the values, send a `POST` request to Cumulocity IoT that changes the value for the `minimum_wait_time_secs` key. See "Configuration" on page 95 for detailed information.

The scale of appropriate values varies depending on what the magnitude of the input value is. If **Output Threshold** is not set, then the block only generates new outputs if it receives an input (this may be appropriate if it is receiving frequent inputs on the value, or if the **Sample** input is being used).

- **Ignore Timestamp** parameter

  For some inputs, the block by default uses the source timestamp available on the input (for example, for a Cumulocity IoT measurement, event or alarm). The block reorders the input based on the timestamp (see also "Input blocks and event timing" on page 73), but drops events that are delayed by too much. If this behavior is not desirable (for example, if a device's clock is not well synchronized, or if data from a device may be delayed), then you can disable this behavior by selecting the **Ignore Timestamp** parameter. If this is selected, the timestamp of the data is ignored, and the model processes the input data as soon as it is received, regardless of what timestamp it has. This may give different results compared to the default behavior of using timestamps. The behavior which is most desirable will depend on the nature of the device and its connectivity to Cumulocity IoT.

  Note that when a model is running in simulation mode, the setting of the **Ignore Timestamp** parameter is ignored. The block will always use the source timestamp, so that when replaying simulation events, the data is guaranteed to be processed in order and this will yield more realistic results (and there is no record of when the data was received, only the source timestamp). See also "About simulation mode" on page 88.

Input blocks and event timing

Input blocks make data from external sources (such as Cumulocity IoT measurements) available to the model. Many data sources have timestamps on each piece of data, which reports the time that a measurement or event actually occurred. There may be delays in transmitting the data to the Apama system for processing, leading to events being received by Apama out of order.

Data sources with timestamps, such as measurements, can be reordered. Operations, for example, do not have timestamps and are therefore processed as they are received, without reordering.

Apama Analytics Builder delivers several input block types which consume data sources with timestamps. These block types provide an **Ignore Timestamp** parameter which allows you to disable data reordering and thus to process the inputs as they are received. See also "Common block inputs and parameters" on page 72.

The following table lists the available input block types and indicates whether they are able to reorder the input:

Input block type	Reordering is possible
Measurement Input	Yes
Event Input	Yes
Alarm Input	Yes
Operation Input	No
Managed Object Input	No

For data sources that have timestamps associated with a piece of data, the input block can handle events received out of order. In order to do this, the input blocks hold all received events in a reorder buffer and delay processing them until a predefined delay time after their source timestamp. By delaying the processing of the event relative to the source timestamp, the input block allows events to be reordered. The key parameter to this process is the amount of time by which the events are delayed. To configure the time in seconds by which the input blocks delay inputs, send a `POST` request to Cumulocity IoT that changes the value for the `timedelay_secs` key. See "Configuration" on page 95 for detailed information.

The input blocks assume that while events may be delivered out of order, they are received by Apama within the defined time delay value. If an event is received after a delay of more than the defined number of seconds (that is, the difference between the timestamp in the event and the time on the system running Apama), then it is dropped. Thus, if the time delay value is set too low, then a small delay may result in Apama dropping an event, which can lead to erroneous results. The higher the time delay value is, the larger is the delay before an event is processed. Thus, it is important to pick a suitable value for the time delay to match the environment for events being delivered into Apama.

The correlator logs the number of dropped events periodically to the correlator log file. See "Configuration" on page 95 for configuring logging throttling and "Accessing the correlator log" on page 99.

Output blocks and event timing

Output blocks make data (such as Cumulocity IoT measurements or operations) from the model available to external systems (such as Cumulocity IoT). Outputs blocks can either produce synchronous or asynchronous values.

The values from an output block which generates synchronous output (such as measurements) can also be consumed by another model in a time-synchronous manner and can be processed by the model with any other data from the same timestamp. See also "Connections between models" on page 82.

The values from an output block which generates asynchronous output can also be consumed by another model, but only in a time-asynchronous manner when data is received back from the external system.

The following table lists the available output block types and indicates whether the output is synchronous or asynchronous:

Output block type	Type of output
Measurement Output	Synchronous
Event Output	Synchronous
Alarm Output	Synchronous
Operation Output	Asynchronous
Managed Object Output	Asynchronous

Fragment properties on wires

Each wire has a primary value that is of the type of the wire: one of `float`, `boolean`, `string` or `pulse`.

In addition to this, some blocks may provide other *fragments* of information alongside the value. These are named properties on the value. They may be other pieces of information provided from an input block, such as the unit in which a measurement is measured, or some extra contextual information for a data source.

Most blocks only operate on the primary value from their input wires, but some blocks can make use of these fragment properties values and extract them into separate output ports (for an example, see the **Extract Property** block). This gives more flexibility in processing more complex data from external sources.

7 Models and Devices

Model execution for different devices

Models are executed independently for different devices. That is, models for one device can execute in parallel, making use of hardware parallelism where possible, if models are processing data (such as `Measurement`, `Event`, or `Operation` objects) for different devices. When defining a model, you can configure it to use data from only one device, or from a number of devices, with each device being handled independently.

Each model must either:

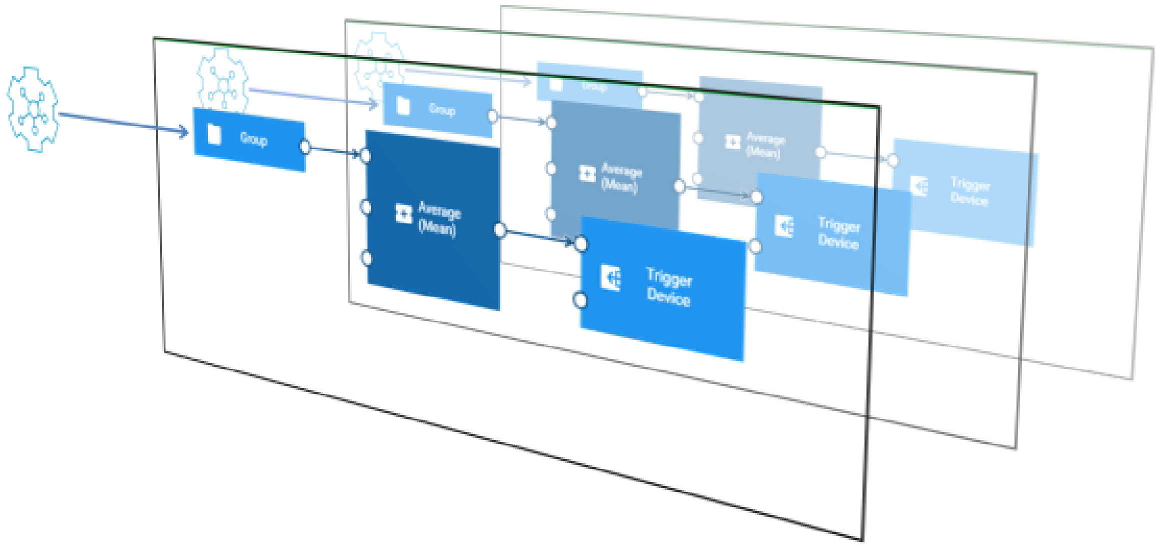
- receive input from a single device, or
- receive input for each device within device groups.

  A device group is a means of organizing devices. A device group can contain any number of devices or other device groups. When a model uses a device group, the model will act on all devices referred to by the device group, either directly or indirectly through members of the device group that are themselves device groups and have device members (or even "grand-children" device group members). A device can be a member of zero, one or many device groups. See also the information on the Device Management application and grouping devices in the *User guide* at "https://www.cumulocity.com/guides/".

  Note: A model that acts on a device group only determines the device group membership when the model is activated. If the membership of a device group changes while a model is running, the model will not behave any differently for any new or removed members of the group. If a device group membership is changed, then models that refer to that device group should be de-activated and re-activated.

It is not possible to mix the two types of input blocks above (but see "Broadcast devices" on page 80). However, data from a model processing single devices can be sent to and received from other models, including models for device groups, and vice versa (see "Connections between models" on page 82).

When a model consumes data from groups, the model behaves as if multiple instances of the model are running, as illustrated below, each one processing data from each device independently. Each instance processes data for a different device, but all share the same blocks and block parameters. The values of the wires will be independent for different instances. Any blocks that are stateful, such as the **Average (Mean)** block, will operate independently of the data from other devices. As with single-device models, if any block causes a runtime error or exception, then the entire model will go into a failed state - it will stop processing data for all devices.



Typically, when using device groups for inputs, all input blocks would use the same group. It is possible to use different device groups. If there are devices in one group but not in another, those blocks will never generate a signal for devices that are not in that group. For some blocks, such as the **Expression** block, this is not useful - an **Expression** block will only generate an output if all of the required inputs have received a value, but it may be useful for `pulse` inputs of a **Gate** block.

When a model has inputs that are consuming data from a single device, then the output blocks generating synchronous outputs must specify the same device. A model can use any number of output blocks generating asynchronous output, all specifying the same device as input device or different devices. Asynchronous output blocks can also be combined with synchronous output blocks.

When a model has inputs that are consuming data from a device group, all synchronous output blocks must specify the **Trigger Device**, a special device in the **Output** category of the palette. The **Trigger Device** generates data (`Measurement`, `Event` or `Operation`) for whichever device that model instance applies to - or whichever device sent the data to trigger that model instance. Asynchronous output blocks in such models can specify the **Trigger Device** or any other specific device.

You can use the model editor to change input and output blocks from one device or device group to another. When changing between a device group and a device, output blocks will switch between the **Trigger Device** and the device specified, so that the model is kept in a usable state. See also "Replacing devices, device groups and assets" on page 50.

The test and simulation modes are only permitted for models using a single device. If you wish to test or simulate a model using a device group, then use the model editor to modify it to apply to a single device within the device group, and then activate the model in test or simulation mode. See "Deploying a model" on page 35 for more information on these modes.

Configuring the concurrency level

By default, the Apama Analytics Builder runtime uses 1 CPU core to execute models. If you want to change the number of CPU cores, send a `POST` request to Cumulocity IoT that changes the value for the `numWorkerThreads` key. See "Configuration" on page 95 for detailed information.

Typically, this configuration value would be set to the number of CPU cores available for the system, but it may be useful to configure this either higher or lower according to what resources are available. It does not need to scale to the number of devices (that is, it is quite reasonable to have 4 worker threads with hundreds of devices, assuming a moderate event rate per device).

Models consuming data from more than one device

Because each device is processed independently, as noted above, it is only possible to use data from, or generate data for, a single device within each model. However, it is possible to configure the concurrency level to 1, at which point it is possible to activate models which can consume data and generate data across different devices. As the concurrency level (`numWorkerThreads`) value is a global setting, this means that for a given Apama Analytics Builder environment, it is only possible to achieve either scale up or models that operate across devices.

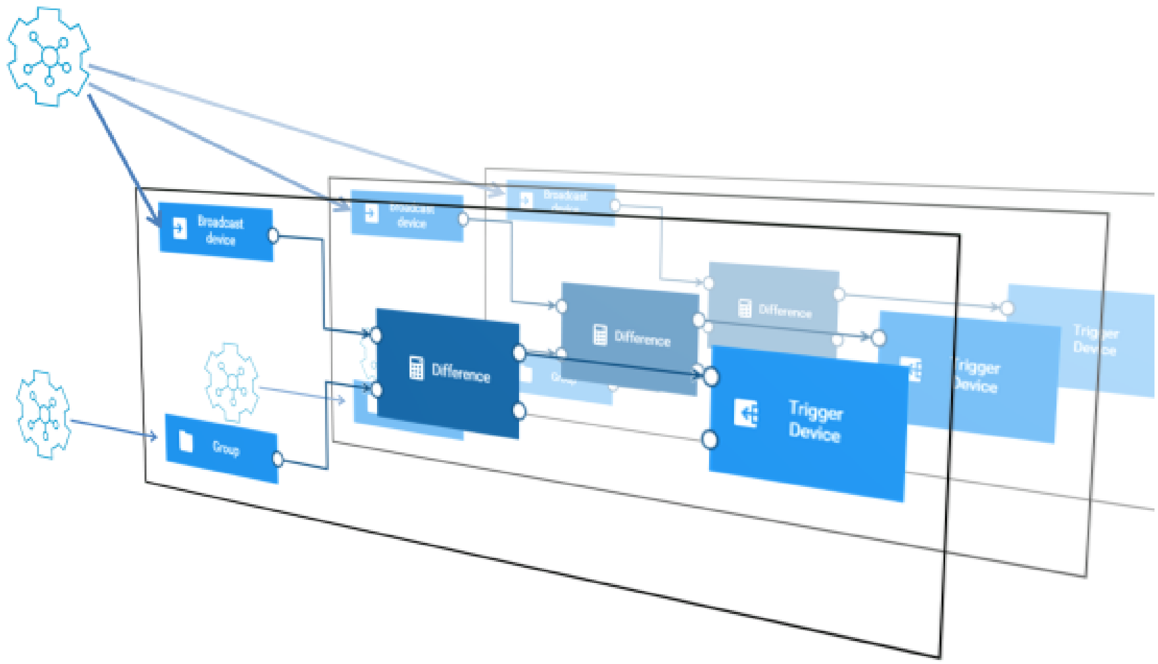
With the concurrency level set to 1, it is still possible to create models which use device groups as inputs, but these continue to operate independently for each device within the device group, and it is still not possible to mix device group and single device input or output.

See also "Broadcast devices" on page 80.

Broadcast devices

While Apama Analytics Builder models typically process each device independently, it is sometimes useful to have signals that apply to all devices. These may be signals from devices, or from other systems that are presented as if they were signals from a device. Apama Analytics Builder thus supports devices that are referred to as *broadcast devices* and signals from these devices are available to all models across all devices.

Broadcast devices can be used as inputs in any model, together with either single device inputs or device group inputs. The diagram below illustrates how a broadcast device applies to all devices within a device group. It is possible to combine signals from devices in a device group with signals from a broadcast device by providing them as different inputs into a processing block such as the **Expression** block.



Unlike other devices, a broadcast device can only be used for synchronous output of a model that only consumes data from broadcast devices. Broadcast output of the asynchronous type can be generated by a model consuming non-broadcast inputs.

It is also not possible to connect models together using synchronous data from a broadcast device output (that is, no model may use a measurement from a broadcast device that is the output of a different model). Models can be connected together using asynchronous outputs from a broadcast device (that is, models may use an operation from a broadcast device that is the output of a different model).

Identifying broadcast devices

Broadcast devices are identified by the presence of a property on the device object in the inventory for that device; this is the `pas_broadcastDevice` property. Thus, whether a device is considered a broadcast device or not is global for that device across all models. It is not permitted to use a device group that contains a broadcast device.

Virtual devices

A virtual device is used when a model is deployed in test or simulation mode. See also "Deploying a model" on page 35.

Virtual devices are objects in the Cumulocity IoT inventory with a `c8y_VirtualDevice` property. This property refers to the identifier of the real device of which the virtual device is a copy.

Use the `creationDate` to find out what `virtualDevice` was created for a model activation and which measurements have that device as their source.

By default, the virtual devices are kept for 30 days. If you want to change this default, you need to change the tenant options. That is, you need to send a `POST /tenant/options` request. For detailed information, see the information on tenants in the *Reference guide* at "https://www.cumulocity.com/guides/". For example, specify the following to set the retention period for the virtual devices to 1 day:

```
{
    "category": "analytics.builder",
    "key": "retention.virtualDevicesMaxDays",
    "value": "1"
}
```

See also "Configuration" on page 95.

Virtual devices are not shown in the Device Management application. Use REST operations as described in the *Reference guide* at "https://www.cumulocity.com/guides/" to find these entries.

Connections between models

You can connect multiple models together using output blocks and input blocks. A model that contains an output block such as **Measurement Output** (for `Measurement` objects of Cumulocity IoT) will generate a series of events, and this can be consumed by a suitable input block (such as **Measurement Input**) in another model.

Note: The events from one model can only be consumed by another model when all involved models are deployed in production mode. When the models are deployed in test or simulation mode, virtual devices are used and the events from one model can therefore not be consumed by another model.

Input and output blocks identify a series of events by specifying a *key* for the series of events. This key can be made up of multiple block parameters, and identifies that series of events distinct from other series of events through the same block type. For example:

- For `Measurement` object input and outputs, the key would be the device, the fragment, and the series. The **Unit** parameter specified in an output block is not considered part of the key (it is for information only) and is not required to match the parameters of the measurement input block.
- For `Event` objects, the key would be the device and the event type.

When one model has a synchronous output block generating a series of events for a given key and a second model has an input block consuming from that same series of events (that is, with the same key parameters), then this forms a connection from the first model to the second. When the first model triggers the output block, this causes the second model to be evaluated with a new input on its input block.

It is also possible to form connections between models using the output from an asynchronous output block. In this case, when the first model triggers the asynchronous output block, the output is generated and sent to the external system (such as

Cumulocity IoT). The data is received back from the external system at some later point in time and causes the evaluation of any other models consuming the data.

Similar to the processing order of wires within a model (see also "Processing order of wires" on page 70), the following applies when an output block in one model generates a series of events that an input block in another model consumes:

- A single model can send the same events to more than one other model. This means, it is possible to have a single model perform some common pre-processing, such as unit conversion or calculating an average (with the **Average (Mean)** block), and that value to be used by multiple other models.
- Models are executed in order with respect to the connections between models formed using synchronous output so that the source of a connection is always evaluated before the target of a connection. If a model has connections from multiple blocks all triggered from the same initial event, then they will all evaluate first, and the receiving model will evaluate with all of the inputs once.

  Connections formed using asynchronous output do not have a specific execution order. A model consuming the output is executed only when the output is received back from the external system.

Similar to the wire restrictions within a model (see also "Wire restrictions" on page 71), there are restrictions on how output blocks and input blocks can be used to connect models together:

- One block across all models is permitted to generate a series of synchronous events for a given key. Multiple output blocks generating asynchronous events can be used within a single model or across multiple models.
- No cycles can be created between models using synchronous output. A model that receives events via an input block synchronously generated from another model cannot include an output block that generates synchronous events that the other model would consume. This applies even if one of the models contains two separate parts, such that there is no actual cycle in terms of wires and connections between models. Cycles among models can be created because of asynchronous outputs. Therefore, care must be taken not to introduce indefinite cyclic executions of models.

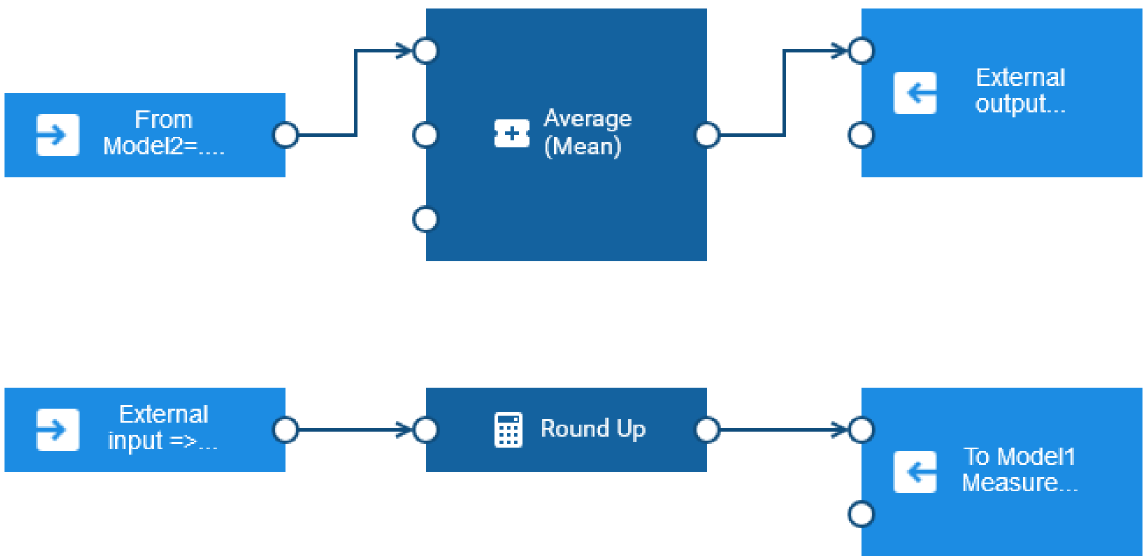
Any model that does not meet these restrictions when used in combination with the already activated models will cause an error on trying to activate it. This will count for the last element in a cycle of models. For such errors, the problem may be in interactions between models rather than a problem specific to a single model, but existing models that have already been activated will not automatically be deactivated. For example, if multiple models all generate the same series of synchronous events (with the same key), then the first model to be activated can be deployed, but all subsequent models will report an error upon trying to activate them.

For example, there are three models: Model1, Model2, and Model3. A cycle may exist if:

- An output block of Model1 produces a series of synchronous events that is consumed by an input block in Model2, and Model2 contains an output block that generates a second series of synchronous events, and

- Model3 contains an input block that consumes a series of events from Model2, and Model3 also contains an output block that generates a series of synchronous events used by an input block in Model1.

Note that only activating any two of these models can be done without error. If activated in order, only Model3 would have an error. But if Model1 or Model2 were deactivated, then Model3 could be activated. The error will occur even if one of the models does not contain a link from the input block that is part of the chain to the output block that forms part of the chain, such as the example for Model3 below: the events from Model2 do not form a cycle to the **To Model1 Measurement** output block, but they count as a cycle as they are both in the same model. (In this case, the issue could be resolved by splitting that model into two models, thus removing the cycle).



Configuring the number of shown devices, device groups and/or assets

By default, a maximum of 10 devices and 10 device groups is shown in the **Input** and **Output** categories of the palette (see also "Adding a block" on page 43). When you use the search boxes that are available for these categories, this default also applies to the maximum number of assets that are shown in the search result. When you click **Load more**, up to 10 more devices, device groups and/or assets are shown.

The same default value is used in the block parameter editor when you select a different device, device group or asset (see "Editing the parameters of a block" on page 44) and when you replace devices, device groups or assets (see "Replacing devices, device groups and assets" on page 50).

If you want to change this default value (to show either more or less items), you need to change the tenant options. That is, you need to send a `POST /tenant/options` request.

For detailed information, see the information on tenants in the *Reference guide* at "https://www.cumulocity.com/guides/".

For example, specify the following to set the value to 20:

```
{
    "category": "analytics.builder",
    "key": "c8yAnalyticsBlocks.queryInventoryPageSize",
    "value": "20"
}
```

See also "Configuration" on page 95.

Searching for input and output assets

By default, only devices and device groups are shown in the palette (see also "Adding a block" on page 43). However, when you use the search boxes that are available for the input and output blocks, all assets (not only the devices and device groups) in the Cumulocity IoT inventory which match the search criteria are shown. You can thus build analytic models by defining any assets in the inventory as input blocks or output blocks.

If you want to restrict the search to show only assets of a specific type (for example, to show only devices), you need to change the tenant options. That is, you need to send a `POST /tenant/options` request. For detailed information, see the information on tenants in the *Reference guide* at "https://www.cumulocity.com/guides/".

For example, specify the following if you only want to show devices:

```
{
    "category": "analytics.builder",
    "key": "c8yAnalyticsBlocks.queryInventoryNameSearchAdditionalFilter",
    "value": "has(c8y_IsDevice)"
}
```

The `c8y_IsDevice` in the value is a so-called *fragment*. You can specify any fragment that is known to Cumulocity IoT, including any fragments that you have created yourself.

You can combine several values. For example, specify the following if you only want to show devices and device groups:

```
{
    "category": "analytics.builder",
    "key": "c8yAnalyticsBlocks.queryInventoryNameSearchAdditionalFilter",
    "value": "has(c8y_IsDevice) or has(c8y_IsDeviceGroup)"
}
```

The default value of this tenant option is `not has(c8y_IsVirtualDevice)`. As long as you do not change this tenant option, virtual devices are not shown as they would not make sense in an analytic model. If you change the value for this tenant option, make sure to specify all assets that you want to see in the search result.

The tenant options are also used in the block parameter editor when you select a different device (see "Editing the parameters of a block" on page 44) and when you replace devices (see "Replacing devices, device groups and assets" on page 50).

See also "Configuration" on page 95.

8 Model Simulation

About simulation mode

You can deploy a model in simulation mode to run it against historical input data (such as Cumulocity IoT measurements). This allows testing the behavior of a newly developed model against historical data or fine-tuning an existing model. Or it allows testing a model against a set of historical data with known properties.

You use the model manager to deploy a model in simulation mode. See "Deploying a model" on page 35 for more details.

Note: Simulation mode is only permitted for models using a single device. If you wish to simulate a model using a device group, then use the model editor to modify it to apply to a single device within the device group, and then activate the model in simulation mode.

When a model is deployed in simulation mode, it uses data from a virtual device (see also "Virtual devices" on page 81). Thus, a simulated model can run alongside other non-simulated models without interfering with them.

A simulated model runs as if it is running at the time of the historical data. The input data are processed in the order of their historical time. The simulated model also uses the historical time for the timestamps of the generated output.

Events, alarms and operations are created with a timestamp. However, with time there can be updates to these objects. For example, an alarm can be cleared or the status of an operation can be changed. As a history of changes to event, alarm and operation objects is not maintained, the object is only replayed at its initial timestamp, with the latest version of its properties. Thus, changes to these objects are not replayed and simulation mode is of limited use if your models depend on changes to objects.

Note: Simulation mode is not permitted for models with input blocks of type **Managed Object Input**.

When running a simulation, historical data is replayed into the Apama correlator from the Cumulocity IoT database. If there is a significant delay in the data being queried from the database or high load in the system, this can lead to dropping the input in exceptional circumstances. A simulated model processes input data at normal speed. For example, if the historical data entries are separated by one second, they are processed one second apart. This means that simulating a model with one hour of historical data will take approximately one hour of simulation time.

Simulation parameters

To deploy a model in simulation mode, you have to provide values for two parameters in the model manager: start time and end time. These values determine the time range for which historical data is to be sent into the simulated model.

- **Start time**

  The start time from which historical data is to be sent into the model.

- **End time**

  The end time until which historical data is to be sent into the model.

Sending of data into the simulated model is stopped when all historical data from the specified time range has been sent.

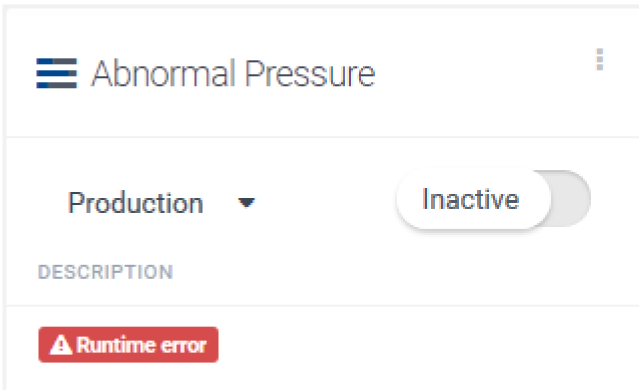
Monitoring dropped inputs

The simulated model may drop delayed input events in exceptional cases. The number of input events dropped across all the models is exposed as a user-defined status with the name `user-analytics-oldEventsDropped`. See also "Monitoring dropped events" on page 95.

9 Monitoring and Configuration

Monitoring

You can monitor the current status of each model in the model manager. The card for a model shows the current mode for this model (such as production mode) and whether it is in the Active (deployed) or Inactive state.



If a model failed to deploy or failed while running, 

is shown on the card for the model. To find out whether a model has failed while processing data, reload all models in the model manager to show their latest states. See also "Reloading all models" on page 38.

Monitoring periodic status

In addition to the status that is shown on the card for a model, it is possible to enable generation of periodic status published as Cumulocity IoT operations or events. See "Configuration" on page 95 on setting the `status_device_name` and `status_period_secs` tenant options.

Each operation has the following parameters:

Parameter	Description
`models_running`	Information about deployed models that are running.
`models_failed`	Information about deployed models that have failed.
`apama_status`	The Apama correlator status metrics. Many status names correspond to the key names in the Apama REST API. The values are returned by the `getValues()` action of the `com.apama.correlator.EngineStatus` event and exposed via the REST API.

Model status

The following information is published for each deployed model that is currently running or has failed:

Name	Description
`mode`	The mode of the deployed model. It is `SIMULATION` for models deployed in simulation mode. Otherwise, it is `PRODUCTION`.
`modeProperties`	Any mode-specific properties of the model. This includes the start and end time of the simulation for models running in the `SIMULATION` mode.
`numModelEvaluations`	The total number of times the model has been evaluated since it was deployed.
`numBlockEvaluations`	The total number of times that the blocks have been evaluated in the model since it was deployed. This is the sum of the count of evaluation for each block in the model.
`avgBlockEvaluations`	The average number of blocks that have been evaluated per model evaluation.
`numOutputGenerated`	The total number of outputs generated by the model since it was deployed.

This information about each model provides insight into the performance or working of models. For example, a model with a much larger number of `numBlockEvaluations` than another model might indicate that it is consuming most resources even though it might have low `numModelEvaluations`. Similarly, it can be used to find out whether a model is producing output at the expected rate relative to the number of times it is evaluated.

The following is an example of the status operation data that is published by Cumulocity IoT:


```
{
    "creationTime": "2018-07-23T21:48:54.620+02:00",
    "deviceId": "6518",
    "deviceName": "apama_status",
    "id": "8579",
    "self": "https://myown.iot.com/devicecontrol/operations/8579",
    "status": "PENDING",
    "models_running": {
        "Package Tracking": {
            "mode": "SIMULATION",
            "modeProperties":{"startTime":1533160604, "endTime":1533160614},
```

```
            "numModelEvaluations": 68,
            "numBlockEvaluations": 967,
            "avgBlockEvaluations": 14.2,
            "numOutputGenerated": 50
        }
    },
    "models_failed": {
        "Build Pipeline ": {
            "mode": "PRODUCTION",
            "numModelEvaluations": 214,
            "numBlockEvaluations": 671,
            "avgBlockEvaluations": 3.13,
            "numOutputGenerated": 4
        }
    },
    "apama_status": {
        "user-analytics-oldEventsDropped": "1",
        "numJavaApplications": "1",
        "numMonitors": "27",
        "user-httpServer.eventsTowardsHost": "1646",
        "numFastTracked": "183",
        "user-httpServer.authenticationFailures": "4",
        "numContexts": "5",
        "slowestReceiverQueueSize": "0",
        "numQueuedFastTrack": "0",
        "mostBackedUpInputContext": "<none>",
        "user-httpServer.failedRequests": "4",
        "slowestReceiver": "<none>",
        "numInputQueuedInput": "0",
        "user-httpServer.staticFileRequests": "0",
        "numReceived": "1690",
        "user-httpServer.failedRequests.marginal": "1",
        "numEmits": "1687",
        "numOutEventsUnAcked": "1",
        "user-httpServer.authenticationFailures.marginal": "1",
        "user-httpServer.status": "ONLINE",
        "numProcesses": "48",
        "numEventTypes": "228",
        "virtualMemorySize": "3177968",
        "numQueuedInput": "0",
        "numConsumers": "3",
        "numOutEventsQueued": "1",
        "uptime": "1383561",
        "numListeners": "207",
        "numOutEventsSent": "1686",
        "mostBackedUpICQueueSize": "0",
        "numSnapshots": "0",
        "mostBackedUpICLatency": "0",
        "numProcessed": "1940",
        "numSubListeners": "207"
    }
}
```


You can monitor the status using the Apama REST API or the Management interface which is an EPL plug-in. See the following topics in the Apama product documentation for further information:

- "Managing and Monitoring over REST" in *Deploying and Managing Apama Applications*, and
- "Using the Management interface" in *Developing Apama Applications*.

Monitoring dropped events

When a model receives an event, it may be dropped if the correlator delivers or processes it too late. See "Input blocks and event timing" on page 73.

The total number of dropped events across all models is periodically published as part of the status operation. The count of the number of dropped events is available as a user-defined status value with the name `user-analytics-oldEventsDropped` in the `apama_status` parameter of the status operation. See also "Monitoring periodic status" on page 92 for details about the operation.

All dropped input events are also sent to channel `AnalyticsDroppedEvents`, allowing you to implement your own monitoring of the dropped events. A dropped input event sent to the channel `AnalyticsDroppedEvents` is packaged inside an event of type `apama.analyticsbuilder.DroppedEvent`. This allows you to extract the original dropped event and perform any analysis on it, for example, categorizing the number of dropped events per device. This can be achieved by writing EPL that listens for the `DroppedEvent` events, aggregates by device identifier and/or time, and sends measurements to Cumulocity IoT that can be monitored. See the *Streaming analytics guide* at "http://cumulocity.com/guides/" for information on how to deploy EPL applications.

Monitoring the model life-cycle

Life-cycle messages are written to the correlator log whenever a model is created or deleted, or when it fails. The log messages may look as follows:

```
Model "Build Pipeline" with PRODUCTION mode has started.

Model "Build Pipeline" with PRODUCTION mode has ended.

Model "Build Pipeline" with PRODUCTION mode has failed with an error:
IllegalArgumentException - Error while validating parameters for the
block "toggle" of type "apama.analyticskit.blocks.core.Toggle":
The "Set Delay" must be finite and positive: -1.
```

See "Accessing the correlator log" on page 99 for information on where to find the correlator log.

Configuration

You can customize the settings of Apama Analytics Builder, the so-called "tenant options", by sending REST requests to Cumulocity IoT. The key names that you can use with the REST requests are listed in the topics below. A category name is needed along with the key name; this is always `analytics.builder`.

You can find some concrete examples in "Using curl commands for setting various tenant options" on page 98. However, you can use any tool you like.

Caution: After you have changed a tenant option using a REST request, the correlator will automatically restart. An alarm with a MAJOR severity will be created in this case; you can view it on the **Alarms** page of the Cockpit application (see the *User guide* at "https://www.cumulocity.com/guides/" for information on how to work with alarms).

Keys for status reporting

Key name	Description
`status_device_name`	The name of the Cumulocity IoT device to which the status operations are to be published. The default name is `apama_status`.
`status_period_secs`	The frequency in seconds at which the status is to be published. The default value is 0 seconds, meaning that status reporting is disabled. You can enable status reporting by setting the frequency to a positive value.
`status_send_type`	How the status is to be published. The default value is `OPERATION`, meaning that the status is published as a Cumulocity IoT operation. You can change this to `EVENT` to publish the status as a Cumulocity IoT event.
`status_event_type`	The event type if the status is published as a Cumulocity IoT event. The default type is `apama_status`.
`status_event_text`	The event text if the status is published as a Cumulocity IoT event. The default text is `Apama Status`.

Keys for model timeouts

Key name	Description
`timedelay_secs`	The maximum delay in seconds before the input block considers an event to be old. The default value is 1 second. See also "Input blocks and event timing" on page 73.

Key name	Description
`logging_throttle_secs`	Logging throttling in seconds. Periodic log messages (for example, those reporting changes in the number of events being dropped by the input block) will not appear more frequently than defined by this constant. The default value is 1 second. See also "Input blocks and event timing" on page 73.
`minimum_wait_time_secs`	The minimum wait time in seconds. Some blocks can generate output automatically, based on the rate of change of the output. This sets a lower limit on the time between such outputs. See also "Common block inputs and parameters" on page 72.
`default_timeout_secs`	The default timeout in seconds for simple request responses. This is used, for example, in requests to Cumulocity IoT. The default value is 10 seconds.

Other keys

Key name	Description
`numWorkerThreads`	The number of worker threads. The default value is 1. See also "Configuring the concurrency level" on page 80.
`retention.virtualDevicesMaxDays`	The retention period in days for keeping virtual devices. The default value is 30 days. See also "Virtual devices" on page 81.
`c8yAnalyticsBlocks.queryInventoryPageSize`	The number of devices, device groups or assets that are shown in the palette, and also in the block parameter editor when you select a different device, device group or assets. The default value is 10 (that is, 10 devices and 10 device groups, or 10 assets). See also "Configuring the number of shown devices, device groups and/or assets" on page 84.
`c8yAnalyticsBlocks.queryInventoryNameSearchAdditionalFilter`	The assets that are shown in the palette when you use the search box, and also in the block parameter editor. See also "Searching for input and output assets" on page 85.

Logged tenant options

The values for some of the tenant options are logged. These are the following:

- `status_device_name`
- `status_period_secs`
- `timedelay_secs`
- `numWorkerThreads`

If you want to find out which values are currently used for these tenant options, you can look them up in the correlator log. See also "Accessing the correlator log" on page 99.

Using curl commands for setting various tenant options

You can set or change various tenant options by sending `POST` requests to Cumulocity IoT. This topic explains how you can do this using the curl command-line tool. See "https://curl.haxx.se/" for detailed information on curl. See also the information on tenants in the *Reference guide* at "https://www.cumulocity.com/guides/".

The syntax of the curl command depends on the environment in which you are working. The syntax for a Bash UNIX shell, for example, is as follows:

```
curl --user username -X POST -H 'Content-Type: application/json' -d
'{"category": "analytics.builder", "key": "keyname", "value": "value"}'
-k https://hostname/tenant/options
```

where:

- *`username`* is the name of a user who has the permission to change the tenant options in Cumulocity IoT. curl will prompt for a password. Or you can provide a password in the *`username`* argument by appending it with a colon (:) and the password. For example:

  ```
  --user User123:secretpw
  ```

- *`keyname`* is one of the keys listed in the previous topics.
- *`value`* is the value that is to be set for the key, which can be a number or a string, depending on the key.
- *`hostname`* is the host name of your tenant where your user application is deployed.
- The `category` is always `analytics.builder`.

For example (Bash shell):

```
curl --user User123 -X POST -H 'Content-Type: application/json' -d
'{"category": "analytics.builder", "key": "numWorkerThreads", "value":
"4"}' -k https://mytenant/tenant/options
```

Accessing the correlator log

The location of the correlator log depends on the environment in which you are working:

- Cumulocity IoT Core:

  The correlator log is accessible via Cumulocity IoT's Administration application. You can find it on the **Logs** tab of the Apama-ctrl microservice. You have to subscribe to the microservice so that you can see the logs. For more information on microservices and log files, see the *User guide* at “https://www.cumulocity.com/guides/”.

- Cumulocity IoT Edge:

  See the *Cumulocity IoT Edge* guide at “https://www.cumulocity.com/guides/” for detailed information on the log file location.