Using Analytics Builder for Cumulocity IoT

Version 10.15.0

November 2022

APAMA

This document applies to Analytics Builder for Cumulocity IoT 10.15.0 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: PAB-UG-10150-20221114

Table of Contents

About this Guide

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

Documentation Roadmap

This documentation is provided in the following formats:

- HTML
- PDF

In addition, reference information for the pre-built blocks (block descriptions, parameters, input port details, and output port details) is available which can be accessed from the model editor, which is one of the tools of Analytics Builder for Cumulocity IoT.

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.
|	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

Product Documentation

You can find the product documentation on our documentation website at https://documentation.softwareag.com.

In addition, you can also access the cloud product documentation via https://www.softwareag.cloud. Navigate to the desired product and then, depending on your solution, go to "Developer Center", "User Center" or "Documentation".

Product Training

You can find helpful product training material on our Learning Portal at https://knowledge.softwareag.com.

Tech Community

You can collaborate with Software AG experts on our Tech Community website at https://techcommunity.softwareag.com. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software AG news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at https://github.com/softwareag and https://hub.docker.com/publishers/softwareag and discover additional Software AG resources.

Product Support

Support for Software AG products is provided to licensed customers via our Empower Portal at https://empower.softwareag.com. Many services on this portal require that you have an account. If you do not yet have one, you can request it at https://empower.softwareag.com/register. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

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 10.15.0

- Additional Analytics Builder samples are now available in the model editor. The provided samples now cover all of the smart rules from Cumulocity IoT.
- Detailed descriptions have been added for the Analytics Builder samples. You can see the first lines of such a description in a tooltip when you move the mouse over the card for a sample, or you can see the full description when you view a sample (or create a model from the sample) and then open the Model Configuration dialog box.
- It is now possible to search for Analytics Builder samples with specific words in their names and to filter them by tags.
- In the previous version, the template parameter names in the Analytics Builder samples were only available in English. As of this version, translations are available for the supported languages.
- The Alarm Output and Measurement Output blocks now have a new **Params Fragment** parameter and a new **Properties** input port. These can be used to set custom properties, similar to the already existing functionality of the Event Output block.
- The Alarm Output block now has a new **Create Asynchronous Output** parameter. It is not selected by default which reflects the block's previous behavior. When selected, alarms can now be created and cleared asynchronously.

  CAUTION:
  Special care must be taken when using the **Create Asynchronous Output** parameter. This may result in infinite loops during model execution (for the same model or across multiple models), leading to performance degradation or the application running out of memory.
- Analytics Builder now supports the `globalTitle` branding option for the category names in the model editor. See also Branding and language customization in the Cumulocity IoT documentation.
- As announced in "What's new in version 10.14.0" on page 12, the `frgment` property has been removed in 10.15. If you have not already done so, you must now change `frgment` to `fragment` in all of your blocks (for example, the value of the **Property Path** parameter in the Extract Property block).

What's new in version 10.14.0

- Samples are now provided in the model manager. These are similar to the smart rules in Cumulocity IoT. The samples are intended to help you get started with creating your own models. You can use a sample as a basis for further development by creating a new model from the sample. You can then edit the new model according to your requirements and deploy it. See also "First steps: Creating a model from a sample" on page 30 and the description of the new **Samples** tab in "The model manager user interface" on page 46.

- The Measurement Output block now has a new **Time** input port. This can be used to set the timestamp of the measurement and works similar to the already existing **Time** input ports of the Alarm Output and Event Output blocks.
- The Measurement Input block sends the property name that is used for a fragment as part of its output value. In previous versions, this property name was `frgment`. This has now been changed to `fragment`. Currently, both property names `frgment` and `fragment` are supported. `frgment`, however, is now deprecated and will be removed in a future release. Therefore it is recommended that you now change `frgment` to `fragment` in all of your blocks (for example, the value of the **Property Path** parameter in the Extract Property block).
- The Measurement Input block now also sends a new property named `value` as part of its output value. This allows other blocks, such as the Text Substitution block, to use the measurement value (for example, to use it in an alarm text).

Known issues

- The template parameter names in the samples are currently only available in English. It is planned to translate them into the supported languages in a later release.

What's new in version 10.13.0

- The Send Email and Send SMS blocks can now be used in simulation and test mode. When running in these modes, these blocks log the output instead of sending an email or SMS.

  In previous versions, when using these blocks in simulation or test mode, the activation of the model failed with the following error message: `Model produces output for device '' not declared by any block. Make sure that device parameters of all blocks are properly tagged.`
- The Text Substitution block now supports the new `FORMAT` parameter for specifying a different time format. For example, `#{time:TZ="America/New_York",FORMAT="HH:mm:ssZ"}` specifies the time zone for New York and the format to be `HH:mm:ssZ`. For a list of valid time format strings, see Format specification for the TimeFormat functions in the Apama documentation.

What's new in version 10.11.0

This version of Analytics Builder does not contain any new functionality.

What's new in version 10.10.0

- An issue which prevented connecting `any` type outputs to blocks which take `float` as input has been fixed. The Expression block has also been updated to handle `any` type inputs.
- The Pulse block now has a new **Mode** parameter. Its default reflects the block's previous behavior: it sends a pulse when the value changes. Using this new parameter, you can now configure the block to send a pulse on every input or on every non-zero value. See also "Type conversions" on page 95.

- The Text Substitution block now supports time zone values. You can now specify a `:TZ=timezone` suffix to use a different time zone, for example, `#{time:TZ=America/New_York}` to use the time zone for New York.
- The **Property Name** parameter of the Extract Property block has been renamed to **Property Path** as it also accepts a path. This is only a label change. You do not need to change existing models. They will continue to work as before.

What's new in version 10.9.0

- A new application called “Streaming Analytics” is now available in the Cumulocity IoT application switcher. Analytics Builder is now part of this new Streaming Analytics application, together with EPL Apps. See also “Starting Analytics Builder” on page 26.
- The following new blocks are now available:
  - Smart rule behavior:
    - KPI (this block implements the core part of the "On measurement threshold create alarm" smart rule; see also Smart rules collection in the Cumulocity IoT *User guide*)
    - Text Substitution (see also Smart rule variables in the Cumulocity IoT *User guide*)
  - Position and geofence support:
    - Position Input (new block type for the input blocks)
    - Geofence
  - Non-device inputs and outputs (see also the "On alarm send SMS" and "On alarm send email" smart rules under Smart rules collection in the Cumulocity IoT *User guide*):
    - Constant Value (see also the list of new input and output ports on existing blocks below)
    - Cron Timer
    - Send SMS
    - Send Email
  - More calculations and aggregates:
    - From Base N
    - To Base N
    - Limit
    - Range
    - Discrete Statistics (see also “Details of Values and Blocks” on page 105)
    - Group Statistics
  - Utilities and flow manipulation to give more flexibility in writing models:

- Duration
- Set Properties (see also the list of new input and output ports on existing blocks below)
- Selector
- Switch (this block is intended to be used with template parameters)

See the "Block Reference" on page 149 for detailed information on the new blocks.

- Additions and changes to existing blocks:
  - Managed Object Input:
    - The **Property Name** parameter is now optional. If not set, any update to a managed object generates an output.
  - Managed Object Output:
    - The **Property Name** parameter is now optional, allowing several properties to be updated in one go (for example, using the new **Set Properties** block).
  - Event Output:
    - New **Time** input port.
    - New **Properties** input port (see also the new **Set Properties** block).
  - Operation Output:
    - New **Properties** input port (see also the new **Set Properties** block).
  - Alarm Output:
    - The **Severity** parameter is now optional. The severity of the alarm can now be set either via the **Severity** parameter or the new **Severity** input port. One of the input port and parameter must be connected or set.
    - The **Severity** input port (but not parameter) also supports a CLEARED value to clear the alarm instead of creating or changing the severity of an existing alarm.
    - The **Message** parameter is now optional. The message of the alarm can now be set either via the **Message** parameter or the new **Message** input port. One of the input port and parameter must be connected or set.
    - New **Time** input port.
    - New **Clear Alarm** input port.
  - Extract Property:
    - The **Property Type** parameter now supports a new option: **Properties**. This is intended to be used with the following block types: **Operation Output** (the **Properties** input port), **Event Output** (the **Properties** input port), and **Managed Object Output** (the **Value** input port).

    - The **Property Name** parameter is now optional so that the **Properties Type** is able to just pass through the properties (see also the new **Set Properties** block).

  See the "Block Reference" on page 149 for detailed information on the above changes.

- The input blocks are now able to distinguish between create and update events. For this purpose, a new **Notification Mode** parameter is now available in the block parameter editor. This is helpful, for example, if you are only interested in new alarms being created and you are not interested in models being activated when the same alarm is updated. This new parameter is available for the input blocks of type **Alarm Input**, **Event Input**, **Operation Input**, and also for the new **Position Input** type. It is not available for input blocks of type **Measurement Input** as measurements cannot be updated. See the "Input" on page 152 category in the Block Reference for more information.
- A new `chain_diagnostics` parameter for monitoring periodic status is now available. It provides diagnostic information about model chains, such as the time when the chain was created or the number of times the chain has been evaluated. See "Monitoring periodic status" on page 138 for more information.
- A new `block_promise_timeout_secs` key for configuring model timeouts is now available. This tenant option defines the timeout in seconds to wait for `Promise` values returned by block actions like `$validate` to be completed. The default is 60 seconds. See "Keys for model timeouts" on page 145 for more information.
- Objects containing the `c8y_Kpi` property are now treated as broadcast devices in addition to `pas_broadcastDevice`. See "Broadcast devices" on page 124 for more information.
- Analytics Builder now supports the standard user interface languages of Cumulocity IoT. See Available languages in the Cumulocity IoT documentation for more information.
- Detailed documentation is now available which covers the distinctions between value types representing continuous-time and discrete-time values, and the `pulse` type. It also covers some of the details of block implementations around windowing and when blocks generate output. See "Details of Values and Blocks" on page 105.

What's new in version 10.7.0

- It is now possible to use template models. A template model is a model in which one or more template parameters are defined. Template parameters can be bound to any number of block parameters, provided that the type of the block parameter is the same as that of the template parameter.

  The following enhancements come with this new feature:

    - You can switch a block's parameter to be a template parameter and then assign a new or an existing template parameter. See "Editing the parameters of a block" on page 62 for more information.
    - A new dialog box is available for editing template parameters. See "Managing template parameters" on page 70 for more information.

  - A new instance editor is provided. The instance editor allows you to set up different instances of the same model. The blocks in each instance can then use different values for the template parameters. Each instance can be activated, deactivated, or use different run modes, independently. See "Using the Instance Editor" on page 81 for more information.
  - The model manager uses a new type of card for the template models. See "The model manager user interface" on page 46 for more information.

  See also "Models" on page 36 which explains the two types of models that can now be created: models with template parameters and (as in previous versions) models without template parameters. This topic also explains the relevant roles for the new type of model, which can be the same person or different persons: the model author and the instance maintainer.
- Models can now use multiple devices in a model with the concurrency level set to more than 1. See also "Model execution for different devices" on page 122.
- The handling of old events is more lenient now. Old events are dropped only if an event for the same timestamp or a more recent timestamp has already been processed by the model. This means that an old event might be processed by some models but dropped by other models. See also "Input blocks and event timing" on page 100.
- For slow chains, the following status information is now published within the `apama_status` parameter:
  - `user-analyticsbuilder.slowestChain.models`
  - `user-analyticsbuilder.slowestChain.delaySec`

  In addition, a message is now written to the correlator log if the slowest chain is delayed by more than 1 second. See "Monitoring periodic status" on page 138 for more information.
- Activation messages are now written to the correlator log whenever the activation of a chain is started and completed. See "Monitoring the model life-cycle" on page 143 for more information.
- The version 1 API of the Analytics Builder Block SDK for input and output blocks is deprecated. See the documentation at https://github.com/SoftwareAG/apama-analytics-builder-block-sdk for information on how to migrate your custom input and output blocks to the version 2 API.

  Important:

  The new features for multiple devices in a model and more lenient handling of old events are disabled if any input and output blocks using the version 1 API of the Analytics Builder Block SDK are added as extensions.
- A new topic has been added which gives new users a brief overview of how to add a simple model, and how to view its output. See "First steps: Creating your first model" on page 27.
- The Block Reference which is shown in the model editor is now also available from the documentation. The "Overview of all blocks" on page 150 provides links to the descriptions of all the blocks in the block reference.

- Analytics Builder now supports branding. This means that the branding settings as described in the Cumulocity IoT *User guide* at https://cumulocity.com/guides/users-guide/enterprise-edition/#branding now also apply for Analytics Builder.
- The appearance of several toolbar buttons in the model editor has changed. Especially the following buttons look different now: the button for changing the devices and the button for showing or hiding the grid.

What's new in version 10.6.6

- The input block type **Managed Object Input** is now able to output the initial value of the property that is specified by the **Property Name** parameter when the model is activated. For this purpose, the **Capture Start Value** 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 more information.
- The **Expression** block now allows integers in intermediate results, for example, as the result of a `floor()` or `round()` function. If the result of an expression is an `integer` value, it is converted to a `float` automatically (there might be a loss of precision). Previously, the **Expression** block could handle only `float`, `string` and `boolean` types. See the Block Reference in the model editor for more information.

What's new in version 10.6

- The following new blocks are now available in the model editor:
  - **Machine Learning** - Invokes the specified Machine Learning model that scores the input data. To use this block, the Machine Learning application needs to be available with the respective Machine Learning models in the tenant.
  - **Counter** - Gives a count of the total inputs and repeated inputs.

  See the Block Reference in the model editor for more information on the new functionality.
- The **Event Output** block type now has an additional input port named **Text Input**. If connected, this sets the text of the event. If not connected, the **Message** parameter determines the text to be used. If neither are set, the model name is used as the text. See the Block Reference in the model editor for more information on the input port.
- Analytics Builder now also supports Polish. See also "Language settings" on page 27.

What's new in version 10.5.7

- The documentation has been updated to mention the permissions that are required to use Analytics Builder in Cumulocity IoT. See "Prerequisites" on page 25.
- The following tenant options can now also be used with measurements: `status_send_type` and `status_event_type`. In addition, a new tenant option `status_send_keys` is now available. See "Keys for status reporting" on page 144.

- The documentation has been updated with the tenant options that are available for simulation mode. See "Keys for simulation mode" on page 146.
- Diagnostic information is now available to users with READ permission for "CEP management". This includes log file contents, copies of EPL applications, and much more. See "Viewing diagnostics information" on page 144.
- Model activations and deactivations are shown in the audit logs of Cumulocity IoT. See "Viewing the audit logs" on page 143.

What's new in version 10.5

- You can now use 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 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 61).
  - 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 62 and "Replacing devices, device groups and assets" on page 69.
  - 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 129).
- 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 101.
- 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 61.
- 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 144.
- You can now set or change various tenant options by sending REST requests to Cumulocity IoT. For detailed information, see "Configuration" on page 144.
- Status reporting is now disabled by default. See "Configuration" on page 144 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 144.
- The default value for the concurrency level (`numWorkerThreads`) has been changed from 4 to 1. See also "Configuring the concurrency level" on page 124.
- It is no longer possible to use the Apama command-line tools to access the correlator.
- Support for creating extensions has been removed from Analytics Builder. To create extensions, use the Analytics Builder Block SDK instead.
- You can now use the 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 Analytics Builder. See also "Creating your own blocks" on page 41.

What's new in version 10.3.2

- 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 61.
- 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 61.

  Note:

  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 128.
- 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 122.
- 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 124.
- In previous versions of 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 122). 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 60.
- 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 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 125.

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 80.
- A German user interface is now available for Analytics Builder. See also "Language settings" on page 27.

2 Getting Started with Analytics Builder

What is Analytics Builder?

Analytics Builder is part of the Streaming Analytics application which runs in Cumulocity IoT, 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 35 for more detailed information.

Analytics Builder consists of the following tools:

- **Model manager.** When you invoke 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 remove them. You can create new models or edit existing models; in this case, the model editor is invoked. Samples are also available which help you get started with creating your own models. See "Using the Model Manager" on page 45 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 57 for detailed information.
- **Instance editor.** If template parameters have been defined in a model, the instance editor lets you set up different instances of the same model which can then be activated and managed separately. The instance editor uses the template parameters that have been defined in the model editor. See "Using the Instance Editor" on page 81 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.

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/.

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 Analytics Builder runtime, and also executes any custom Apama rules added using EPL apps. For more information, see the *Streaming Analytics guide*, available at the above URL.

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 126). 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 analytic model. EPL apps can be written directly in Cumulocity IoT using the Streaming Analytics 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 Analytics Builder implement the logic required; see "Creating your own blocks" on page 41.

Analytics Builder can be used with both Cumulocity IoT Core (cloud) and Cumulocity IoT Edge (local installation). You can customize several aspects of Analytics Builder by setting various tenant options. See "Configuration" on page 144 for detailed information.

Prerequisites

Browsers

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

Permissions

To use Analytics Builder in Cumulocity IoT, you must at least have the following permissions:

Permission type	Permission level
CEP management	ADMIN

Permission type	Permission level
Option management	READ
Inventory	READ

This is typically achieved by using a global role which has those permissions, and where the role has access to the Analytics Builder application. See the information on the Administration application in the *User guide* at https://www.cumulocity.com/guides/ for details on managing permissions.

Microservice

To use Analytics Builder, you need the Apama-ctrl microservice in Cumulocity IoT.

Your tenant may be subscribed to the Apama-ctrl-starter microservice, in which case you are limited to at most 3 active analytic models. Custom blocks written with the Analytics Builder Block SDK cannot be used with Apama-ctrl-starter. Contact Software AG support to discuss adding more capabilities.

If your tenant is subscribed to the Apama-ctrl-smartrules microservice, Analytics Builder is not available.

Starting Analytics Builder

Analytics Builder is part of the Streaming Analytics application in Cumulocity IoT. You start the Streaming Analytics application 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 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.

If Cumulocity IoT or the browser is set to a language that is currently not supported by Analytics Builder, the user interface is shown with the default language, which is English.

First steps: Creating your first model

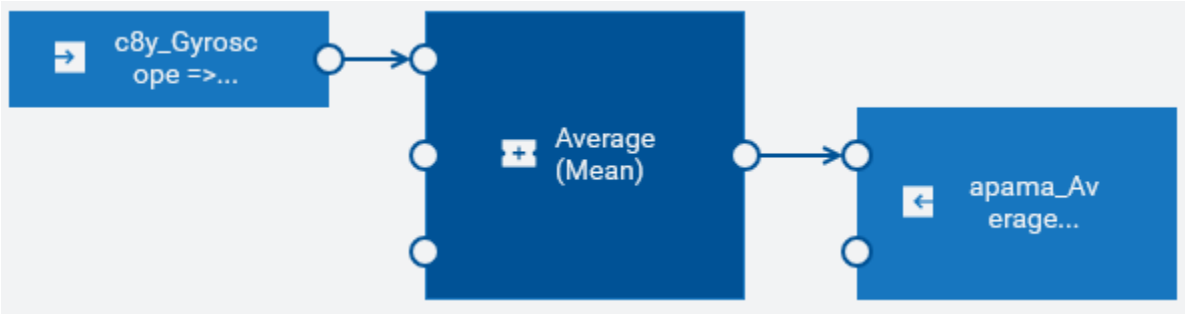
This topic gives a brief overview of how to add and design a new model, and how to view its output. It is not intended to be a comprehensive description of the full range of possibilities provided by Analytics Builder. Therefore, explanations are kept to a minimum. For more detailed information, see the remainder of this documentation.

The steps below require that a device has already been registered in Cumulocity IoT. Preferably, this is a device which is already sending measurement values to Cumulocity IoT. This can be, for example, a smartphone on which the Cumulocity IoT Sensor App has been installed (see Cumulocity IoT Sensor App in the Cumulocity IoT documentation for detailed information).

The model that you add will contain three blocks:

- An input block which receives measurement values from a device.
- A block that calculates the mean of the measurement values over time.
- An output block that sends the calculated mean values to Cumulocity IoT's Device Management application so that they can be viewed there.

When you have completed all steps below, your model will look similar to the following:



Step 1: Log on to Cumulocity IoT and switch to Analytics Builder

You access Cumulocity IoT via a web browser. See Getting Started in the Cumulocity IoT documentation for detailed information.

1. Ask your administrator for the URL that is required to start Cumulocity IoT.
2. Enter your user name and password in the login screen.

3. In the top bar of the Cumulocity IoT user interface, click the application switcher button () and then click **Streaming Analytics**.

4. On the home screen of the Streaming Analytics application, click the **Open** button that is shown below the Analytics Builder heading.

   Or click **Analytics Builder** in the navigator on the left.

   Note:

   If the navigator is currently hidden, click the small arrow at the very left of the top bar to toggle the display of the navigator.

Step 2: Add a new model

The first page that is shown when you invoke Analytics Builder is the model manager.

1. On the **Models** tab, click **New model** in the toolbar.

2. In the resulting dialog box, enter a model name and click **OK**.

Step 3: Add the input block

You design your model in the model editor. The model editor is shown after you have entered the model name. The palette which is shown on the left contains all blocks that can be added to a model. You add a block by dragging it from the palette onto the canvas. The blocks for the input devices that have been registered in Cumulocity IoT are shown under **Input**.

1. Drag the input block for your device onto the canvas.

   The block parameter editor is automatically shown. The block type of an input device is **Measurement Input** by default. We will use this default setting.

   Note:

   If the block parameter editor is not shown (for example, because you clicked an empty space on the canvas after dragging the input block onto the canvas), click the block using the *left* mouse button to show the block parameter editor.

2. Select the fragment and series for which the input block is to listen.

   If the device has previously sent data, the drop-down list box offers one or more values for selection. An example for the Cumulocity IoT Sensor App would be **c8y_Gyroscope => gyroscopeY**.

3. Select the **Ignore Timestamp** check box.

   This makes sure that the measurements are processed in the same order as they are received.

If you need detailed information on the currently selected block, view the block reference in the documentation pane on the right. If the documentation pane is currently not shown, click  .

Step 4: Add the block that calculates the mean of the measurement values

1. In the palette, expand **Aggregates**.
2. Drag the **Average (Mean)** block onto the canvas.
3. In the block parameter editor, specify a value for **Window Duration (secs)**, for example "10".

   The specified number of seconds will be used to control what duration the measurement is averaged over. Smaller values will react quicker to changes in values, larger values will give more smoothing of the value.

Step 5: Add the output block

1. In the palette, expand **Output**.
2. Drag the output block for your device (that is, a block with the same name as your input block) onto the canvas.
3. Specify "apama_Average" as the fragment name.
4. Specify "value" as the series name.

Step 6: Connect the blocks

To pass the values from one block to another, you have to connect the blocks with wires. You attach the wires to the ports, that is, to the small circles that are shown to the left and/or right of a block.

1. Click the **Value** output port of the input block and drag the mouse to the **Value** input port of the **Average (Mean)** block.
2. Click the **Average** output port of the **Average (Mean)** block and drag the mouse to the **Value** input port of the output block.

Step 7: Save the model and go back to the model manager

1. In the toolbar of the model editor, click  to save your newly created model.
2. In the toolbar of the model editor, click  to leave the model editor and thus to return to the model manager.

Note:

Only saved models are listed on the **Models** tab of 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.

Step 8: Activate the model in production mode

A card for the newly added model is shown on the **Models** tab of the model manager. A new model is automatically set to draft mode and inactive state. You will now activate your new model in production mode. This deploys the model so that the measurements from your device are processed.

1. Click the drop-down menu on the card which currently shows **Draft** and select **Production**.
2. Click the toggle button on the card which currently shows **Inactive**. This changes the state to **Active**.

Step 9: Go to the Device Management and view the measurements

To view the measurements that are sent from your active model, you have to switch to Cumulocity IoT's Device Management application. See Device Management in the Cumulocity IoT documentation for detailed information.

1. Click the application switcher button () and then click **Device management**.
2. In the navigator on the left, click **Devices** and then **All devices**.
3. Locate your device and click its name to display the device details.
4. Click **Measurements** on the left. This is a dynamic tab which is only shown when measurements are available for the device.

   The resulting page shows several charts, visualizing the data sent from your device. It should now also show a chart titled "Apama_average" in which you can view the values that are sent from your newly created model. You may have to reload the page to see this new chart. See Measurements in the Cumulocity IoT documentation for more information on the **Measurements** tab.

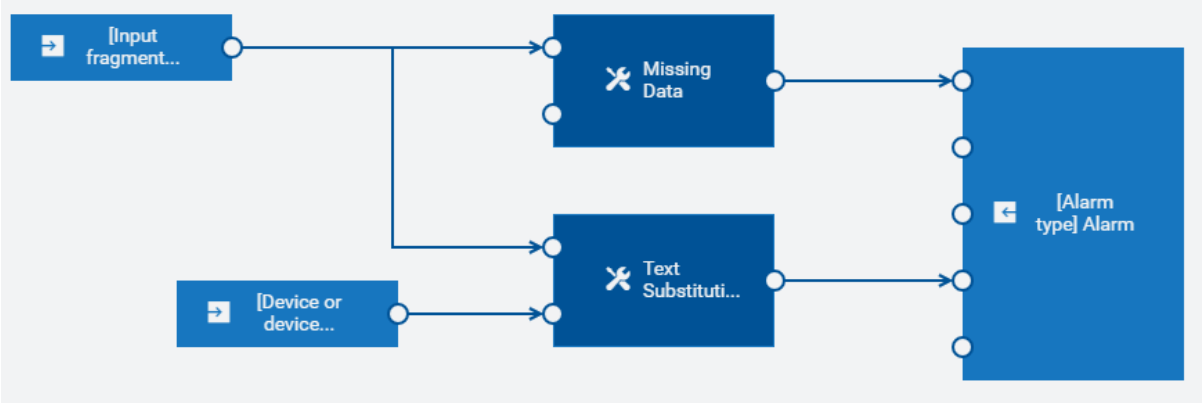
First steps: Creating a model from a sample

This topic gives a brief overview of how to create a model from a sample. It is based on the **On missing measurements create alarm** sample. Your new model will create an alarm if no new measurement data has been received for a specified time period.

This topic is not intended to be a comprehensive description of the full range of possibilities provided by Analytics Builder. Therefore, explanations are kept to a minimum. For more detailed information, see the remainder of this documentation.

The steps below require that a device has already been registered in Cumulocity IoT. Preferably, this is a device which is already sending measurement values to Cumulocity IoT. This can be, for example, a smartphone on which the Cumulocity IoT Sensor App has been installed (see Cumulocity IoT Sensor App in the Cumulocity IoT documentation for detailed information).

The following image shows the blocks that are defined in the **On missing measurements create alarm** sample.



The sample uses predefined template parameters. After you have created a model from the sample, you can create multiple instances of the model and you can specify different values for the template parameters. See also "Models" on page 36 which explains the difference between models without template parameters and models with template parameters.

The following is a brief description of the blocks that are defined within the sample:

- The input starts with the **Measurement Input** block waiting for new incoming measurements that match a given value that is defined with the **Input fragment and series** template parameter. The name of that parameter is the label that you can see on the input block.
- The output from the **Measurement Input** block is then passed to the **Missing Data** block which triggers an output if no input is received within the time defined with the **Duration (seconds)** template parameter.
- The output from the **Missing Data** block is used as the trigger for the **Create Alarm** input port of the **Alarm Output** block. The name of the **Alarm type** template parameter is the label that you can see on the output block.
- The output from the **Measurement Input** block is also passed as input to the **Object** input port of the **Text Substitution** block, along with the input from the **Managed Object Input** block which is passed to the **Source** input port of the **Text Substitution** block. The name of the **Device or device group** template parameter is the label that you can see on the input block.
- The **Text Substitution** block supports replacement of placeholders. For example, if the input text is "Missing measurements of type: #{type}", then the `#{type}` value is replaced by the actual `type` of the measurement. See "Text Substitution" on page 211 for more details.
- The **Text Substitution** block is configured to use the **Alarm text** template parameter as user input. It applies the required substitutions and then sends the string containing the substitutions from its **Output** output port to the **Message** input port of the **Alarm Output** block.
- The **Alarm Output** block requires the **Alarm type** and **Alarm severity** template parameters to be configured and creates an alarm whenever it is triggered by the **Missing Data** block.

Step 1: Create a new model from a sample

The **Samples** tab of the model manager lists all sample models that are provided with Analytics Builder. You can view a sample by simply clicking on its card, but you cannot edit or deploy it. However, you can use the samples as a basis for developing your own models, by creating a model from a sample.

1. Go to the **Samples** tab of the model manager.
2. Click the actions menu of the **On missing measurements create alarm** sample and then click **Create model from sample**.

   The new model is immediately shown in the model editor. It has the same name, description and tags as the sample.
3. If you want to rename the model, click the model name which is shown at the left of the toolbar. You can then specify a new name in the resulting Model Configuration dialog box.
4. In the toolbar of the model editor, click  .
5. In the toolbar of the model editor, click  to leave the model editor and thus to return to the model manager.

   Note:

   Keep in mind that only saved models are listed on the **Models** tab of the model manager.

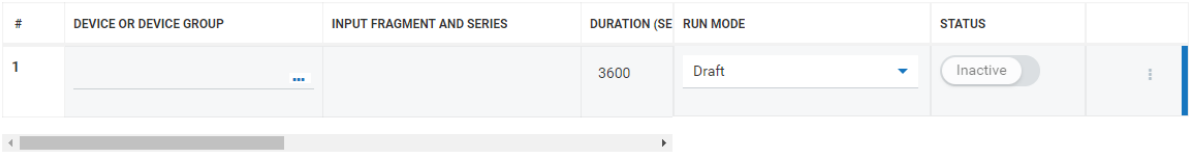
Step 2: Create a new instance of the model

The sample model uses template parameters. So when you turn the sample into a model, you create a so-called template model. You cannot activate a template model directly in the model manager. Instead, you must create at least one instance of the model, and you can then activate that instance using the instance editor.

1. On the **Models** tab of the model manager, locate the card for your newly created model.
2. To invoke the instance editor, click **0 Instances** which is currently shown on the card.
3. Click **New Instance** to create the first instance of your new model.

Step 3: Fill in the template parameter values

In the instance editor, a row is shown for each instance that you create. A column is provided for each template parameter that is defined in the template model, with the name of the template parameter being the column header. As long as an instance is not active, you can adjust the values for that instance.



Use the horizontal scroll bar below the instance table if not all template parameters (columns) are shown on the screen.

1. Click the field below the **Device or device group** column header. In the resulting dialog, select the device or device group that you want to use for this instance and click **Use**.
2. In the text box below the **Input fragment and series** column header, specify the details of the measurement input that you want to monitor in the following format:

   ```
   <valueFragmentName>.<valueSeriesName>
   ```

   For example, if the measurement fragment is `c8y_Gyroscope` and the series is `gyroscopeX`, then you must enter the following:

   ```
   c8y_Gyroscope.gyroscopeX
   ```

   Tip:

   If you want to find out which fragments and series are available to your device, without changing the predefined template parameters of the **Measurement Input** block, go back to the model editor, drag the input block for your device from the palette onto the canvas and open the **Fragment and Series** drop-down list box. This lists all the values that you can use. However, instead of the => that you can see in the drop-down list box, you have to use a dot (.) in this case. Don't forget to remove this block again after you have decided which value to use.
3. The fields below the **Duration (seconds)**, **Alarm type**, **Alarm text** and **Alarm severity** column headers already contain default values (see also the above description of the blocks). Adapt them to your requirements. For example, change the duration to 30 seconds, rename the alarm type to “MyAlarmType”, keep the predefined alarm text, and set the alarm severity to **Minor**.
4. In the toolbar of the instance editor, click **Save**.

Step 4: Activate the instance

You will now activate the instance in production mode. This deploys the instance so that the measurements from your device are processed.

1. In the **Run Mode** column of the instance editor, click the drop-down menu for the instance and select **Production**.
2. In the **Status** column of the instance editor, click the button which currently shows **Inactive** to change the status to **Active**.

Step 5: Send in the data from your device

Once the instance has been activated, send in the data from your device. The instance starts monitoring the device once measurement data starts arriving and creates an alarm if no data is received within the configured duration.

For our example case with the gyroscope measurements from a smartphone, it should be sufficient that you simply turn off the smartphone display while the Cumulocity IoT Sensor App is still running.

Step 6: Go to the Device Management and view the alarms

To view the alarms that are sent from your active instance, you have to switch to Cumulocity IoT's Device Management application. See Device Management in the Cumulocity IoT documentation for detailed information.

1. Click the application switcher button () and then click **Device management**.
2. In the navigator on the left, click **Devices** and then **All devices**.
3. Locate your device and click its name to display the device details.
4. Click **Alarms** on the left.
5. On the resulting page, check the alarms that are sent from your device. If you have edited your instance as described above, you should see a MINOR alarm after 30 seconds, saying "Missing measurements of type: c8y_Gyroscope". See Working with alarms in the Cumulocity IoT documentation for more information on the **Alarms** tab.

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.

You can create two different types of models: models without template parameters and models with template parameters.

Models without template parameters

All blocks in the model use defined input devices or device groups and contain defined parameter values. Such a model can be activated immediately in the model manager.

Models with template parameters

A model in which one or more template parameters are defined is called a “template model”. Template parameters can be bound to any number of block parameters, provided that the type of the block parameter is the same as that of the template parameter.

For example, you can define a template parameter for the device name and another for the threshold value. These template parameters can later be set individually in the different instances of the model. For example, one template parameter can specify a device which can then be used for several input and output blocks. Or one instance can use device ABC with a threshold value of 100, and another instance can use device XYZ with a threshold of 200. Models with template parameters are not activated directly in the model manager. You have to create at least one instance of the model, and you can then activate each instance separately using the instance editor.

The scope of the template parameters is local to the model in which they are defined. In other words, template parameters defined in one model cannot be used in any other model that is deployed in same tenant or subtenant. The names of the template parameters must be unique within the scope of the model in which they are defined.

There are two relevant roles for this type of model, this can be the same person or different persons:

- **Model author**

  The model author creates the model and defines all of its blocks, parameters and wires. Most importantly, the model author creates the template parameters and binds them to the appropriate parameters in selected blocks.

- **Instance maintainer**

  The instance maintainer creates the instances of the model and assigns values to the template parameters that are to be used by each instance.

The model author has the following options to define a template parameter:

- It can have default value which is provided as the default value in the instance editor. The instance maintainer may then leave it at the default value or change it to another value.
- It can be optional. The instance maintainer then has the possibility to either provide a value or leave it blank.
- It can be required. The instance maintainer must then provide a value. A required value is one that is not optional and has no default value.

Template model instances

Template model instances hold the values to be used in models with template parameters (see also "Models" on page 36).

For example, two devices may have similar checks on data from the devices, but use different threshold values for those checks. In this case, you would configure an instance for each of the devices, specifying which device and what threshold to use.

Each instance can be activated, deactivated, or use different run modes, independently.

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 38.
- Output blocks, which send data to external sources. An output block normally represents a device that has been registered in the Cumulocity IoT inventory. But there are also blocks for sending an email or SMS to specified receivers. See also "Output blocks" on page 38.
- Processing blocks, which receive data from the input blocks and send the resulting data to the output blocks. See also "Processing blocks" on page 38.

Note:

For detailed information on each block, see "Overview of all blocks" on page 150 which provides links to the descriptions of all the blocks in the block reference.

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. A parameter can be configured with a value or a template parameter.

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 122.

In addition, 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 124.

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 122.

Other output blocks are **Send Email** and **Send SMS** to send emails and text messages. These blocks depend on the tenant environment being correctly configured to be able to deliver the emails and text messages (see also Providing SMS provider credentials in the Cumulocity IoT documentation). Unlike the other blocks, these are not associated with devices within the Cumulocity IoT platform.

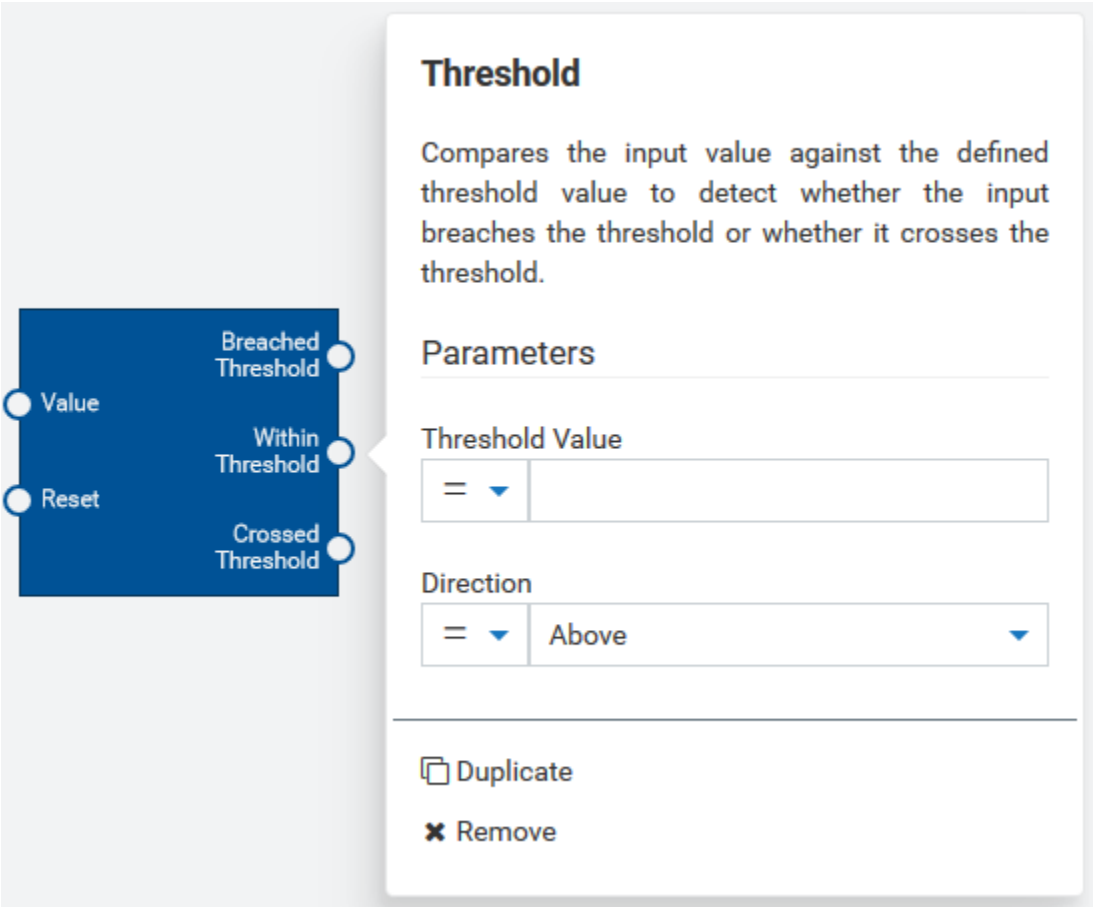
Processing blocks

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.
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.

Creating your own blocks

You can use the Analytics Builder Block SDK to write, test, and package custom blocks and to upload these blocks into 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 --restart
```

To upload an extension, the user specified in the `--username` argument must have CREATE permission for "Inventory" in Cumulocity IoT, in addition to the permissions listed in “Prerequisites” on page 25.

The Apama-ctrl microservice is restarted after running the above command. The user must have the ADMIN permission for "CEP management" to request a restart.

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 89 for detailed information.

Note:

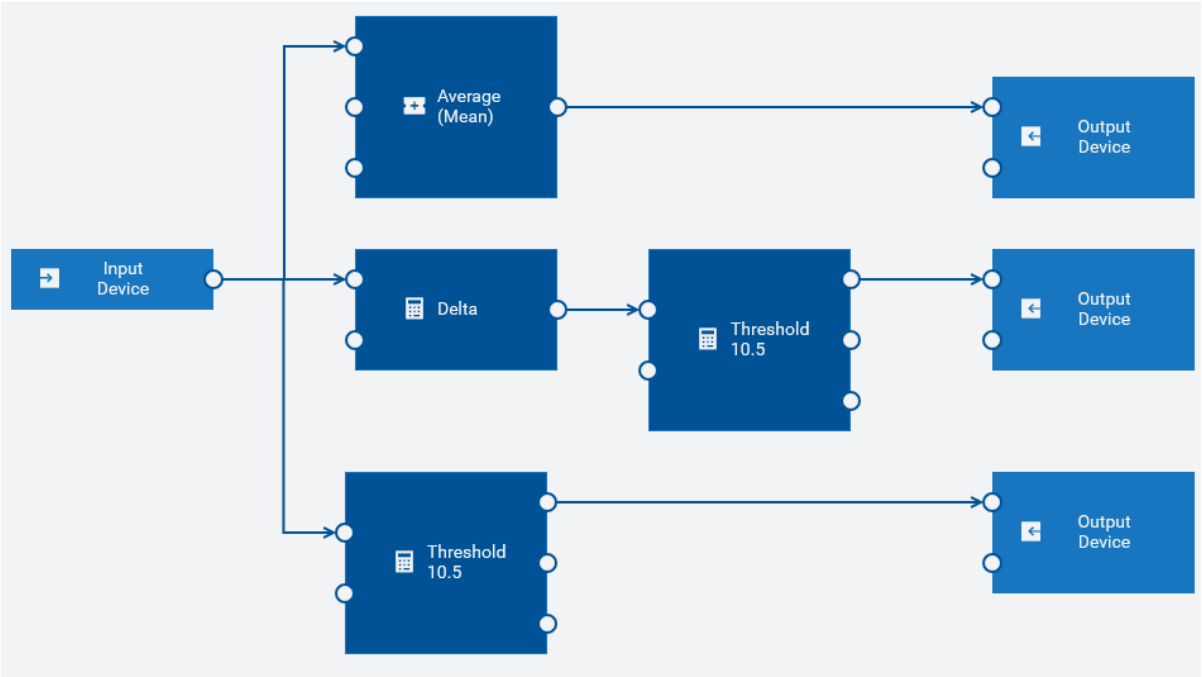
The network of blocks in a model cannot contain any kind of cycles. See “Wire restrictions” on page 98 for more information.

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

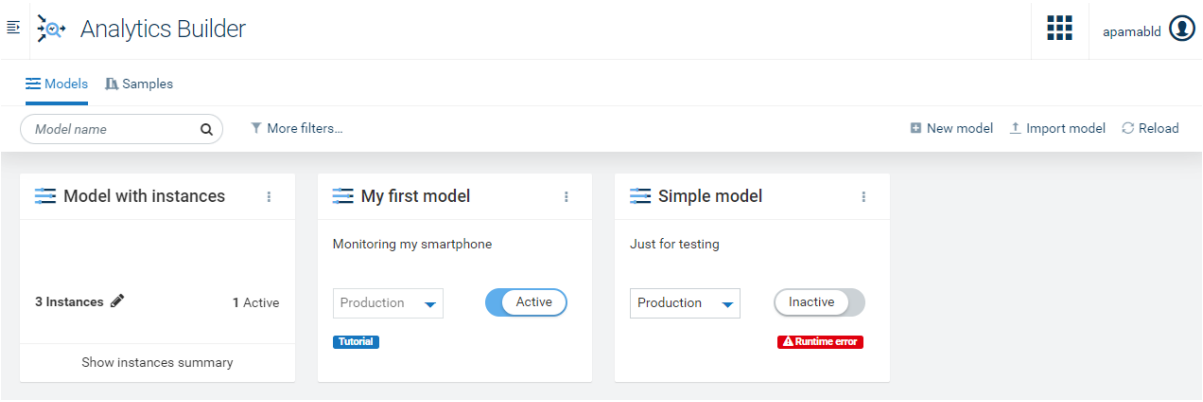
The model manager contains two tabs: the **Models** tab which shows all currently defined models and the **Samples** tab which shows sample models that are intended to help you get started with creating your own models.

Note:

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

The Models tab

The **Models** tab lists all available analytic models within the current Cumulocity IoT environment as cards. You add new models and manage the existing models from here.



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 51). 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 57 for detailed information.

There are two types of models, and the cards for these models look different:

- When a card shows a mode (such as **Draft** or **Production**) and state (**Active** or **Inactive**), it pertains to a model that has no template parameters. Such a model can be activated immediately in the model manager. See "Deploying a model" on page 52 for more information.

  If Runtime error is shown on the card of a deployed model, this model is no longer processing events. Click the error icon to display information on what went wrong.

- When a model has template parameters, it acts as a template. In this case, the number of defined and active instances is shown on the card. A template model is not activated directly in the model manager. Instead, you use the instance editor to create a number of instances, where each instance provides values for the template parameters. Each instance has a mode and can be activated and deactivated in the instance editor, as with models without any template parameters.

To edit the instances, you can simply click the total number of instances (see also "Editing the instances of a model" on page 51). This invokes the instance editor. See "Using the Instance Editor" on page 81 for detailed information.

You can flip the card for a template model to show more details. Click **Show instances summary** to do this. You can then see the number of instances in the different modes.

If an error icon (such as 

) is shown on the card of a template model, at least one of the instances is no longer processing events. Click the error icon to display information on what went wrong.

As long as a model has no template parameters, there will be zero instances and the card shows the controls for selecting a mode and activating it.

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).

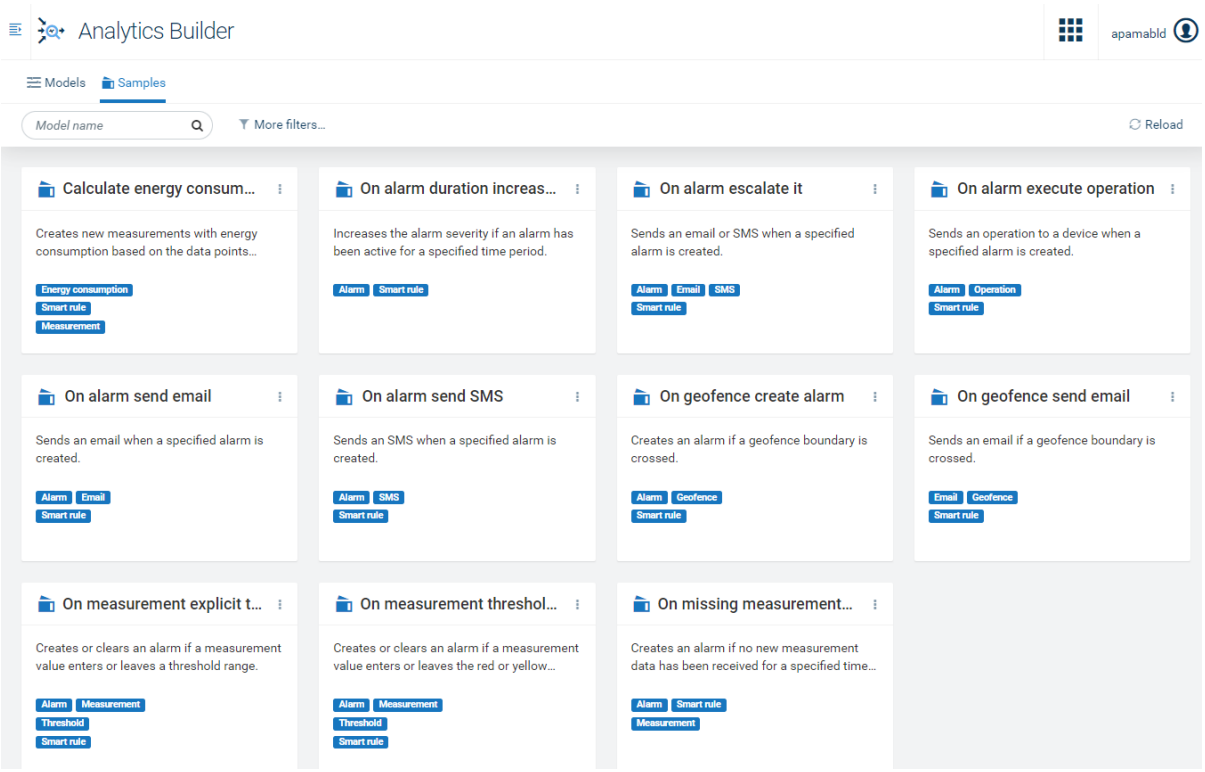
If a description or tags have been defined for the model, this is shown on the card for that model. If you want to change the name, the description or the tags of a model, you have to do this in the model editor. See "Changing the name, description, and tags of a model" on page 59.

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, **Clear search** 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 and samples" on page 48 for detailed information.

The Samples tab

The **Samples** tab lists all sample models that are provided with Analytics Builder as cards.



If the name of a sample or its description is not fully shown on the card, you can move the mouse over the name or description to see the full name or description in a tooltip.

You can view the samples, but you cannot edit or deploy them. To view a sample, you can simply click on the card that is shown for the sample. The model editor is then invoked in read-only mode. See "Viewing a sample" on page 55 for more information.

If you want to use a sample as a basis for further development, you can create a model from the sample. You can then edit the new model according to your requirements and deploy it. See "Creating a model from a sample" on page 55 for more information.

You can easily locate a sample by entering its name or part of the name in the **Model name** search box (for example, "geofence" or "email"). You enter and clear the search criteria in the same way as described above for the **Model name** search box on the **Models** tab. You can also filter the samples by their tags; see "Filtering the models and samples" on page 48 for more information.

Filtering the models and samples

The model manager offers several ways to reduce the number of cards that are shown on the **Models** and **Samples** tabs, thus letting you quickly locate the models or samples that you are looking for.

Filtering also works in combination with a model or sample name that you specify in the **Model name** search box which is explained in "The model manager user interface" on page 46.

To filter the models or samples

1. On the **Models** or **Samples** tab of the model manager, click **More filters** in the toolbar.

2. In the resulting dialog, select one or more filters for the models. For samples, it is only possible to filter by tag.

   On the **Models** tab, 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 50 and "Changing the name, description, and tags of a model" on page 59). Open the **Filter by tag** drop-down list box, select one or more tags, and click **Apply**.

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

   On the **Samples** tab, you can filter the samples by tag only. 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.

3. Click **Apply filters**.

   The toolbar of the **Models** or **Samples** tab now shows the types of filters that are currently applied. This is an example of the **Models** tab:

   Click **Clear filters** in the toolbar if you want to clear these filters. Or to clear a specific filter, click on the × that is shown in a filter icon, or click the filter name in the icon and deselect that filter (and other filters if required) in the resulting dialog box. Clicking **Reset filters** in that dialog box clears all filters.

Adding a new model

When you add a new model, the model editor is invoked. See "Using the Model Editor" on page 57 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 60.

You can also create a new model from a sample. See "Creating a model from a sample" on page 55 for more information.

To add a new model

1. On the **Models** tab of the model manager, click **New model** in the toolbar.

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 and samples" on page 48). 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 59 for a brief overview of how to add blocks and wires to the new model.

Editing an existing model

You can edit (or view) 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

- On the **Models** tab of the model manager, 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).

  Or click the actions menu of the card and then click **Edit**.

  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 57 for further information.

  Note:

  If you do not have sufficient permissions (that is, you only have READ permission for "CEP management" instead of ADMIN permission), the actions menu provides a **View** command instead of the **Edit** command.

Editing the instances of a model

When one or more blocks in a model use template parameters (see also "Managing template parameters" on page 70), you can set up different instances of that model.

Each instance can then use different values for the template parameters and can be activated independently from the other instances. The instances are defined and activated in the instance editor.

Note:

The actions below are only available when template parameters have been defined for the model, that is, when the card for the model shows the number of defined instances.

To edit the instances of a model

- On the **Models** tab of the model manager, click the total number of instances on the front of the card.

  Alternatively, you can also do one the following:

- Click the actions menu of the card and then click **Instances**.
- Or click **Show instances summary** to flip the card and then click the **Edit Instances** button on the back of the card.

  Note:

  Show instances summary is only visible (and thus you can only get to the back of the card) if there are any instances (regardless of state).

This invokes the instance editor. See "Using the Instance Editor" on page 81 for further information.

Deploying a model

A model (or instance) can have one of two states. The current state is always indicated on the card 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.

  Note:

  Test mode is not supported for a model which contains a custom block which consumes input data and also produces output data. Custom blocks are created with the Block SDK; see also "Creating your own blocks" on page 41.

- **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 131.
- **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.

Note:

The above information on the different states and modes similarly applies for the instances of a template model. The following instruction, however, only applies for non-template models. If you want to deploy the instances of a template model, see "Deploying an instance" on page 84.

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

To deploy a model

1. On the **Models** tab of the model manager, click the drop-down menu on the card for 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 132.

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 (that is, deactivate) 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.

Note:

If you want to undeploy the instances of a template model, see "Undeploying an instance" on page 84.

To undeploy a model

- On the **Models** tab of the model manager, click the toggle button on the card for the model that you want to undeploy so that **Inactive** is then shown on the button.

Duplicating a model

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

The duplicated model 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 duplicate is “My Model #1”. The number in the model name is increased by one with each subsequent duplicate that you create. The duplicated model gets the same description as the original model. It is recommended that you edit the duplicate and give the model a meaningful name and description.

To duplicate a model

- On the **Models** tab of the model manager, click the actions menu of the model that you want to duplicate and then click **Duplicate**.

  A card for the duplicated 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

- On the **Models** tab of 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. On the **Models** tab of the model manager, click **Import model** in the toolbar.
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. On the **Models** tab of 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

- On the **Models** tab of the model manager, click **Reload** in the toolbar.

Viewing a sample

The samples are always in read-only mode. You can view the contents of each sample that is currently listed in the model manager.

For example, you can look at the block parameters and view the documentation for each block that is used in the sample. You can navigate and zoom the sample in the same way as a regular model, but you cannot add or edit anything. However, you can create a new model from a sample (see also "Creating a model from a sample" on page 55).

To view a sample

- On the **Samples** tab of the model manager, simply click the card that is shown for the sample.

  Or click the actions menu of the card and then click **View**.

Creating a model from a sample

You can create a new model from each sample that is currently listed in the model manager. The new model gets the same name, description and tags as the sample.

Note:
You must save the new model so that it is listed in the model manager. If a model with that name already exists, you are prompted to save the new model with a different name.

To create a model from a sample

- On the **Samples** tab of the model manager, click the actions menu of the sample from which you want to create a new model and then click **Create model from sample**.

Or, when the sample is currently shown in the model editor, click **Create model from sample** in the toolbar.

The new model is immediately shown in the model editor and you can now change each aspect of the model.

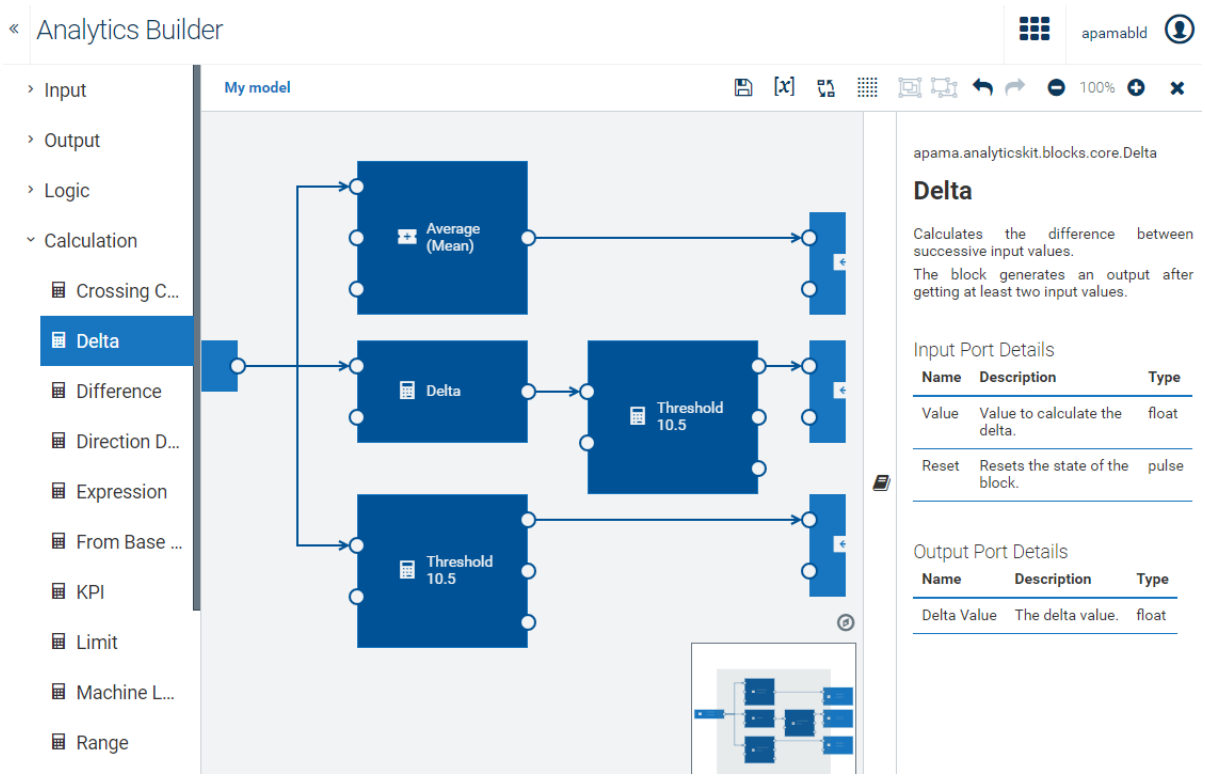
5 Using the Model Editor

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 50 and "Editing an existing model" on page 51.

Note:

The model editor is also invoked when you view a sample. In this case, the model editor is invoked in read-only mode and the palette on the left is not shown. See also "Viewing a sample" on page 55.



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 80).

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 79.

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 65.

CAUTION:

Changes are only saved when you click  (see also "Saving a model" on page 60). 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. On the **Models** tab of the model manager, click **New model**. Enter a model name in the resulting dialog box. See also "Adding a new model" on page 50.
2. In the model editor, drag the required blocks from the palette onto the canvas. See also "Adding a block" on page 61.
3. Refer to the block documentation as necessary. See also "Viewing the documentation for a block" on page 65.
4. Use the block parameter editor to specify the parameters of the block. See also "Editing the parameters of a block" on page 62.
5. Connect the appropriate blocks with wires. See also "Adding a wire between two blocks" on page 67.
6. Save your changes. See also "Saving a model" on page 60.

   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 60.
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 52.

For detailed background information, including restrictions, see "Wires and Blocks" on page 89.

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 and samples" on page 48).

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 65.

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 128.

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 129.

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 66.

   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 73.

   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 62.

   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 not 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 removing the currently selected block. See "Duplicating a block" on page 66 and "Removing a block or wire" on page 68 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 69 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 the input and output blocks, 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 the **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 128.

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 129.

4. It is possible to use a template parameter instead of specifying a value for a block parameter. This allows different values to be used for this block parameter in different instances of the model (see "Using the Instance Editor" on page 81 for more information). Create a template parameter of a matching type in the Template Parameters dialog box (see "Managing template parameters" on page 70), switch the block parameter to use a template parameter (see below) and select the desired template parameter from the drop-down list box. Or create the template parameter directly in the block parameter editor (see below).

   The block parameter editor provides the following options in a drop-down list box:

   - 
     When selected, you can specify a value for this parameter using the adjacent control. This value is validated in the block parameter editor.
   - 
     When selected, you can select a template parameter from the adjacent drop-down list box. You can only select a template parameter that is of the same type as the block parameter to which you want to assign it; template parameters of unsuitable types are not available for selection. Template parameters are not validated in the block parameter editor.

     If you want to add a new template parameter directly in the block parameter editor, type a name in the text box of the above drop-down list box. As soon as you start typing and if a template parameter with that name does not yet exist, the option **Add template parameter *name*** is shown below the text box. Click this option to add the new template parameter and thus make it available in the Template Parameters dialog box. The new template parameter will have the same type, optional and default values as the block parameter. If a template parameter with the name that you are specifying exists already, but with an incompatible type, the name and type is shown below the text box but cannot be selected.

5. 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 remove 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.

6. 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 65.

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 60.

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.

Note:

You can also view the block reference directly in this documentation. See Block Reference.

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 above 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 remove 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 62).

- 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 65. 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 duplicate 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 duplicate.

- Or to duplicate several blocks at the same time, select them as described in "Selecting blocks and wires" on page 65 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 89 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 98 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.

Removing a block or wire

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

To remove a block or wire

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

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

- Or to remove several blocks and/or wires at the same time, select them as described in "Selecting blocks and wires" on page 65 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 removal 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 62 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 128.

   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 129.

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**.

Managing template parameters

A model may specify zero or more template parameters, which can be used in place of defined values for block parameters. For example, instead of defining 100 as the threshold value for a **Threshold** block in the block parameter editor, you can assign a template parameter.

If a model has any template parameters, then multiple instances of the model can be created, and different values can be specified for the template parameters. See also "Models" on page 36 which explains the difference between models without template parameters and models with template parameters, and the different roles: model author and instance maintainer.

If you want to use template parameters, you must first define them in the Template Parameters dialog box as described below. After you defined a template parameter, you can assign it to specific parameters of the same type in the block parameter editor (see also "Editing the parameters of a block" on page 62).

Note:

It is also possible to define template parameters directly in the block parameter editor.

Models with no template parameters can be directly activated in the model manager, with a single instance of the model running. This is different when you save the model after you have assigned at least one template parameter to one or more block parameters, you can no longer activate the model directly in the model manager. Instead, you must create at least one instance of the model, and you then activate that instance using the instance editor. See "Using the Instance Editor" on page 81 for detailed information.

To define the template parameters for the instances of the current model

1. In the toolbar of the model editor, click [x] to invoke the Template Parameters dialog box.

   When at least one template parameter has been defined, a checkmark is shown on the above toolbar icon: [x].

   This dialog box is initially empty and you have to create the template parameters that you want to use in your model. When template parameters have already been defined, they are all shown in this dialog box.

   If you have a long list of template parameters, you can easily locate the template parameter that you are looking for by entering its name or part of the name in the search box. When search criteria are currently applied, an ✖ is shown in the search box; click this to clear the search and thus to show all available template parameters.

   After a template parameter has been assigned to one or more block parameters, the **Usage Count** column indicates the number of places in which the template parameter is used. For example, when a template parameter is used in two places, this means that it has been assigned to two block parameters. These can be within the same block or in different blocks.

2. To create a template parameter, click **Create new template parameter**.

   This adds an empty row at the bottom of the dialog box. You can click **Create new template parameter** several times to add several empty rows that you can fill one after the other.

3. Specify the following information for each template parameter:

   - **Name**. Type a unique name to identify the template parameter within the current model. This name can later be selected in the block parameter editor.
   - **Type**. Select the value type of the template parameter from the drop-down list box. The type can be, for example, a string or float, the name of a device or device group, or the parameter of a specific block.

     An input block specifies a device or a device group, while an output block specifies a device or a trigger device. For template parameters, the same template parameter and thus value can be used for both input and output blocks. If a template parameter is set to refer to a device group, then using it in an output block will be treated as the trigger device. Typically,

a single template parameter would be used for all input and output blocks, and may be a single device or a device group, in which case the block output goes to the device within the group that triggered a model evaluation (so a model calculating an average of a measurement and outputting to a measurement would generate a new measurement for each device independently). Even if a different template parameter whose value refers to a different group was used, the model output would only be sent to the device that triggered a model's evaluation.

- **Optional**. An optional value can remain blank or can be set later by the instance maintainer. When you select this check box, it is not possible to specify a default value.
- **Default Value**. You can only specify a default value when the **Optional** check box is not selected.

  Exception: Boolean types always have a value and cannot be optional. They are "false" by default (that is, the check box for the default value is not selected).

  If you specify a default value, this default value will be provided in the instance editor when the instance maintainer creates a new instance. The instance maintainer can then either leave this default value unmodified or change it as required for that instance.

  When setting the default value for a device, an additional dialog box appears when you click the **Default Value** field. 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 62 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 that you want to use.

Note:

If there is a block parameter for which a required value has not been specified, then the instance cannot be activated. Attempting to do so will report an error.

4. You can update a template parameter at any time. This includes the name, whether it is optional or not, and the default value. All blocks in which the updated template parameter is defined are automatically adapted to use the new values. The only exception is the type. You can only change the type if the template parameter is not used in any block of the model.

5. When the model is inactive, you can reorder the template parameters. This affects the sequence in which they are shown in the instance editor. Drag a row to a different position using the ⇕ control that is shown next to that row. See also "Filtering and sorting the instances" on page 85.

6. You can only remove a template parameter if it is not used in any block of the model. To remove a template parameter, click the actions menu (the three vertical dots at the end of a row) and then click **Remove**.

7. Click **OK** to store the changes in memory and to close the dialog.

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 60.

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 60.

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 51) or a new model that you first have to create (see also "Adding a new model" on page 50).

4. When the model editor is shown, press Ctrl+V to paste the copied items into the model.

Working with groups

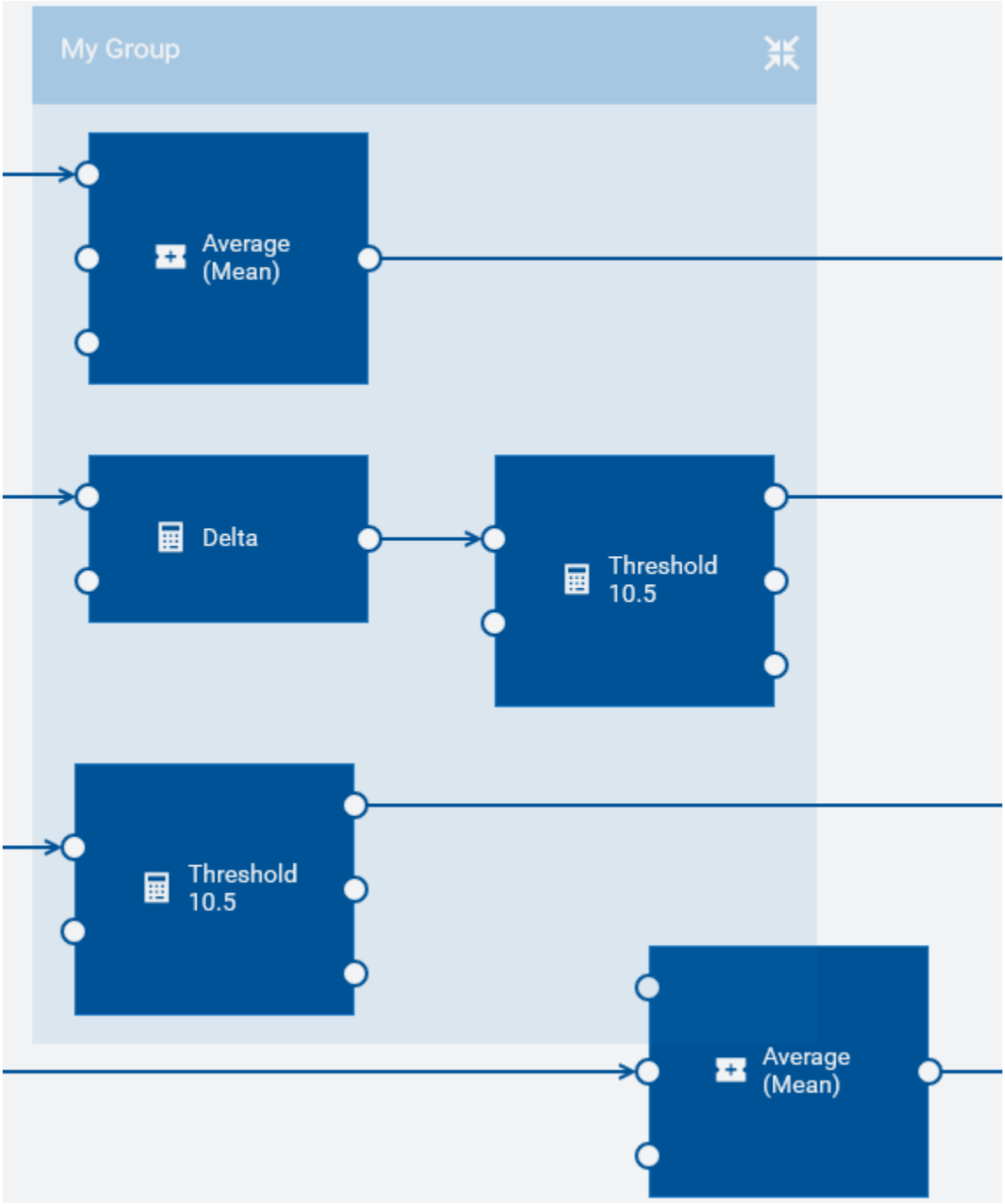
What is a group?

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 61.



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 66). 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 duplicate 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 duplicate 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 duplicate either within the group (this can be the same or a different group) or on the canvas. See also "Duplicating a block" on page 66.

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 61 and "Moving blocks into a group" on page 77.

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 65.

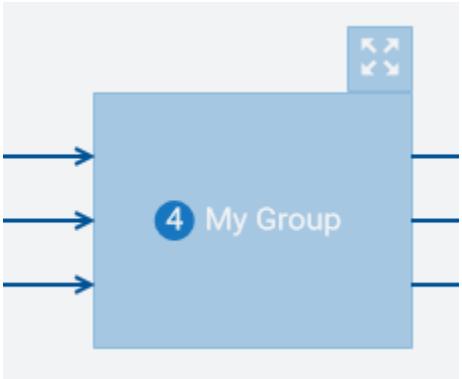
2. In the toolbar of the model editor, click  .

   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 75). 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 61.

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 75.

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 65). 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.

Moving blocks from a group to the canvas

You can move a block from a group to the canvas. All existing wires are retained.

When the last item of a group has been moved to the canvas, the group is automatically removed. If you want to move all items to the canvas at the same time, you can simply ungroup the entire group. See "Ungrouping a group" on page 78.

To move blocks from a group to the canvas

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

Removing blocks and wires from a group

You remove blocks wires from a group in the same way as removing them directly on the canvas. The only prerequisite is that the group is currently expanded. See "Removing a block or wire" on page 68.

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

Duplicating a group

You can duplicate each group that is currently shown on the canvas. The original group and its duplicate 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 duplicated.

You can also copy a group into a different model, see "Copying items to a different model" on page 73.

To duplicate a group

- On the canvas of the model editor, click the group that you want to duplicate (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 duplicated to the position at which you want to place the duplicate.

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.

Removing a group

You can remove each group that is currently shown on the canvas. Wires to blocks outside of the group are removed.

CAUTION:
When you remove a group, all blocks and wires within this group are removed from the model.

To remove a group

- On the canvas of the model editor, click the group that you want to remove (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 79.

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 65.

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

- In the model editor, click the toolbar button for toggling the display of the grid.

  When the grid is hidden, the button looks as follows:  .

  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 Using the Instance Editor

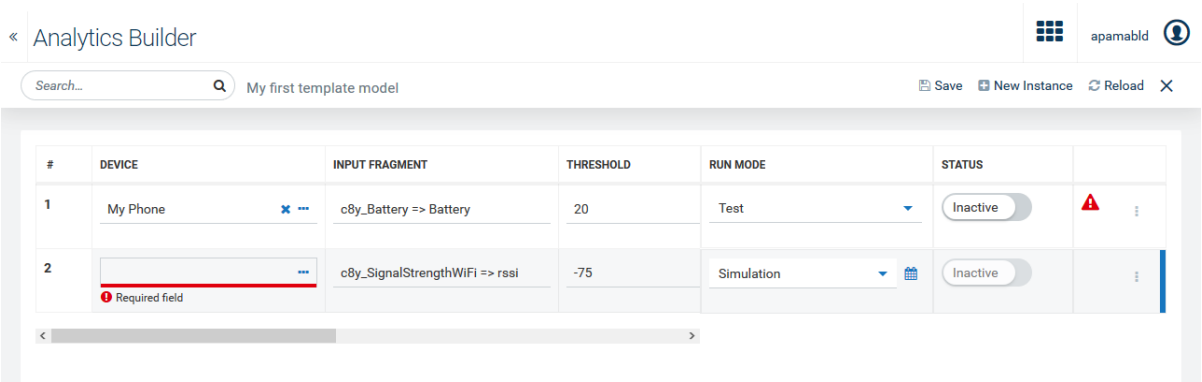
The instance editor user interface

A prerequisite for invoking the instance editor is that one or more template parameters have been defined in the model editor (see also "Managing template parameters" on page 70).

The instance editor allows you to set up different instances of the same model. The blocks in each instance can then use different values for the template parameters, and the instances can be activated independently from the other instances. You invoke the instance editor from the model manager. See also "Editing the instances of a model" on page 51.

Note:
To edit the instances, you must have ADMIN permission for "CEP management". If you have READ permission, you will only be able to view the instances.

The instance editor shows the instances for a selected model. If there are any instances, a table shows the values of the instances, the mode and whether the instance is active.



A row is shown for each instance. A column is provided for each template parameter that is defined in the template model, with the name of the template parameter being the column header. When an instance is not active, you can adjust the values for that instance.

A horizontal scrollbar is available if not all template parameters (columns) can be shown on the screen.

The right side of the table shows the mode and status of each instance. You activate (deploy) and deactivate (undeploy) the instances from here. See also "Deploying an instance" on page 84 and "Undeploying an instance" on page 84.

Each row that is shown for an instance has an actions menu (the three vertical dots that are shown at the very right of the row) which contains commands for managing the instance (for example, to remove the instance).

You can control the list of instances by filtering and sorting. See "Filtering and sorting the instances" on page 85 for more information.

If the error icon is shown near the end of a row, the corresponding instance is no longer processing events. Click that icon to get more information.

When you open the instance editor, it may happen that template parameters have been changed since you last edited the instances and that they no longer use the same values types as before. If the values specified in the instance editor are still compatible, they are converted to the new value types. Incompatible values (including check boxes for boolean types and values that are shown in drop-down list boxes) are automatically removed. Each field from which the value has been removed shows an error underline and a corresponding error message.

Adding an instance

When you add a new instance, a new row is added to the instance editor table. You can then either immediately fill in the required values, or you can first add all required rows and then fill the rows one after the other.

To add an instance

1. In the toolbar of the instance editor, click **New Instance**.

   This adds a new row at the bottom of the table. New instances (rows) are shown with a background color until they have been saved.

2. Fill in the template parameter values, as defined by the model. See also "Editing an instance" on page 83.

Editing an instance

You provide the parameter values for instances in the same way as you provide values for blocks in the model editor (see also "Editing the parameters of a block" on page 62).

The instance editor table provides different types of input controls, depending on the type of template parameter:

- Text boxes are provided in which you can enter values, depending on the setting of the template parameter (for example, a string or a float). Your input is validated as you type. For example, it is not possible to enter a string value in a text box that expects a float value.
- Check boxes are provided for boolean values. Selecting a check box corresponds to setting the value to `true`.
- Drop-down list boxes are provided when you can select a different value (for example, to select a different rule for rounding).
- When you edit a device or device group, an additional dialog box appears. 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 62 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 that you want to use.

Instances (rows) that have been edited but have not yet been saved are shown with a background color until they have been saved.

When a text box requires a value that has not yet been specified, a message is shown, indicating that this is a required field. It is possible to save the instances and leave the instance editor, and set all of the required values at a later point in time. As long as the missing fields of an instance have not been specified, it is not possible to activate that instance.

Deploying an instance

You can activate (that is, deploy) each instance separately. For example, one instance can be in production mode and another in test mode. See "Deploying a model" on page 52 for more information on the different modes; that information applies to both regular models and template models.

When you activate an instance, all changes for that instance are first saved and the instance is then activated.

When an instance is activated, the template parameter values, where supplied, are taken and applied to those block parameters in the model which use a template parameter binding. If no template parameter value is provided, then a default value for that template parameter is used, if there is one. If no template parameter value is supplied in the case of a required template parameter, then the instance will fail to activate.

Once an instance is active, you cannot modify the template parameter values or mode without deactivating the instance first. If any instances are active, then the model is read-only and cannot be modified until all instances are deactivated.

To deploy an instance

1. In the **Run Mode** column of the instance editor, click the drop-down menu for the instance that you want to deploy and select one of **Production**, **Test** or **Simulation**.

   You cannot activate instances that are in draft mode.

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 132.

3. When the toggle button in the **Status** column 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.

   In the case of an error, an  error icon is shown at the right of the table and the instance cannot be activated. Click the error icon to get more information.

Undeploying an instance

You can deactivate (that is, undeploy) each instance that is currently in production, test or simulation mode and for which the toggle button in the **Status** column of the instance editor shows **Active**.

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

To undeploy an instance

- In the **Status** column of the instance editor, click the toggle button for the instance that you want to undeploy so that **Inactive** is then shown on the button.

Filtering and sorting the instances

If you have a long list of instances, you can easily locate the instances that you are looking for by entering a value in the search box. Or you can enter part of the value. This searches all input fields in the instance editor and only lists the instances (rows) that contain this value. All values that match the filter are highlighted. The search criteria are not case-sensitive. When search criteria are currently applied, an ✕ is shown in the search box; click this to clear the search and thus to show all available instances.

You can also sort the columns of the instance editor table. To do so, click the arrows that are shown when you move the mouse to a column header. This sorts the instances according to the values in that column (for example, alphabetically or by number). Clicking again sorts the column in the opposite direction. Fields with required values in that column that have not yet been specified can thus be shown either at the very top or bottom of the column. You can also sort the instances alphabetically according to run mode and status, for example, to show the active instances at the top.

Editing a value will not affect the display of rows in the instance editor table. If you want to reapply the search and sorting, you have to save and reload the instances.

Adding a new instance will not affect the display of rows in the instance editor table. If you add a row after sorting, the row is always added at the bottom of the table, unless you reload the instances.

Note:

You can also reorder the template parameters in the Template Parameters dialog box (see "Managing template parameters" on page 70). This affects the sequence in which they are shown in the instance editor.

Duplicating an instance

You can duplicate each instance (row) that is currently listed in the instance editor. The original instance and its duplicate will then both have the same template parameter values and the same mode. However, the duplicated instance is always inactive even if the original instance is active.

To duplicate an instance

- In the instance editor, click the actions menu of the instance that you want to duplicate and then click **Duplicate**.

A new row for the duplicated instance is immediately shown at the bottom of the instance editor table.

Removing an instance

You can remove each instance that is currently listed in the instance editor. When you remove an instance that is currently deployed, it is first undeployed and then removed.

To remove an instance

1. In the instance editor, click the actions menu of the instance that you want to remove and then click **Remove**.

2. In the resulting dialog box, click **Remove** to confirm the removal.

Saving the instances

You can save the instances even if there are still rows in which required information needs to be specified. This is helpful if you want to add that information at a later point in time.

Note:

When you activate an instance, all of your recent changes are automatically saved. See also "Deploying an instance" on page 84.

To save the instances

- In the toolbar of the instance editor, click **Save**.

  This command is only enabled when changes have been applied to the instances. It saves only those instances where the rows that are highlighted with a background color.

Reloading the instances

You can refresh the display to show the latest state of all instances, or to see whether deployed instances have entered a failed state.

To reload the instances

- In the toolbar of the instance editor, click **Reload**.

  If there are unsaved changes when reloading, you are prompted to save these changes first.

Leaving the instance editor

When you leave the instance editor using the corresponding toolbar button, you are returned to the model manager.

To leave the instance editor

- In the toolbar of the instance editor, click ✕.

  If there are unsaved changes when leaving, you are prompted to save these changes first.

7 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 67.

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 95.
`any`	A value that may be any of the above types. See also "The any type" on page 95.

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

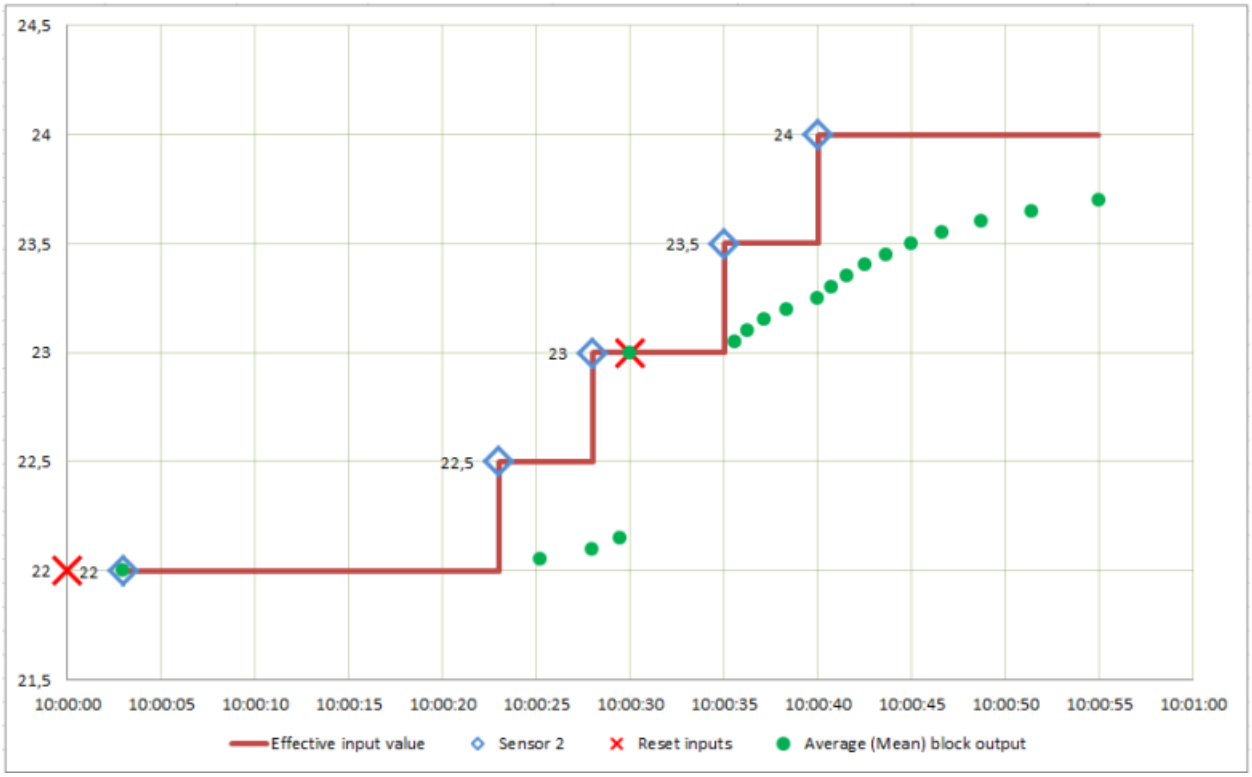
Note that two inputs (to different input ports of the block) to the same block with the same timestamp only generate a single output. For each wire within a model (and each input block), there can only be a single value for a given point in time. An input block cannot generate more than one output for 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 98).

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		
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	

Time	Reset signal	Sensor 2	Effective input value	Average (Mean) block output	Values in window history	Notes
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	
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	✓	✗	permitted if the value is of type float or boolean, other values fail at runtime
	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. This is the default behavior of the **Pulse** block.

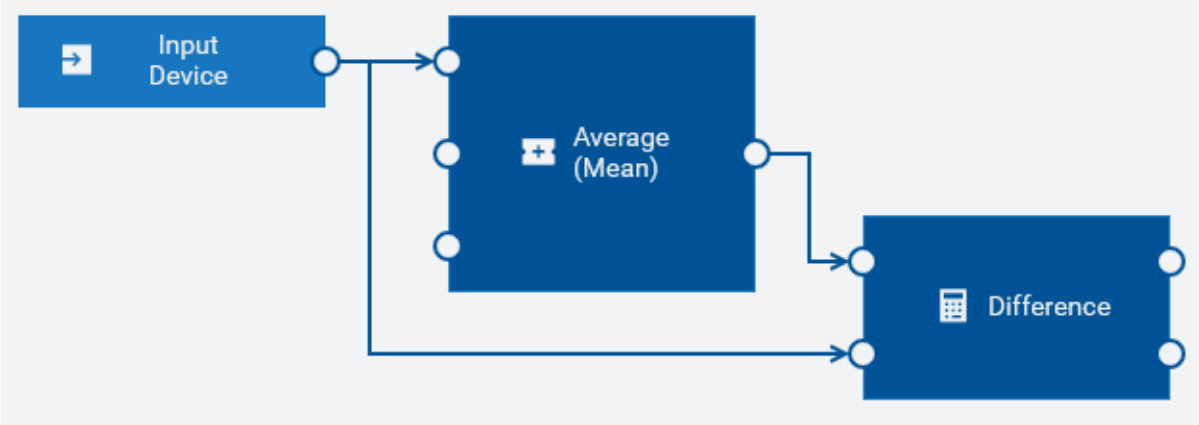
Different types of pulse conversions are possible with the **Pulse** block, depending on the setting of its **Mode** parameter. The conversions in the different modes are described in the table below:

		From block with output type				
		pulse	**boolean**	**float**	**string**	**any**
Connect to Pulse block in mode	**On value change (default)**	✓	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)
	On every input	✓	pulse occurs on every input	pulse occurs on every input	pulse occurs on every input	pulse occurs on every input
	On non-zero values	✓	pulse occurs on every true input	pulse occurs on every non-zero input	pulse occurs on every non-empty input	pulse occurs depending on value's type as described in cells to the left

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 90), 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 97 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 98) 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 90) 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 90 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 144 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 Cumulocity IoT measurements, events and alarms, the block by default uses the source timestamp available on the input. The block reorders the input based on the timestamp (see also "Input blocks and event timing" on page 100), 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 132.

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.

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 98.

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

Input block type	Reordering is possible
Position Input	Yes

Note:

The **Position Input** block is a specialized **Event Input** block. You can also use the **Cron Timer** block to activate a model periodically. Unlike the above blocks, the **Cron Timer** block is not associated with a device and can be found in the **Utility** category of the palette.

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 144 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 if an event for the same timestamp or a more recent timestamp has already been processed by the model. Thus, it is possible that an old event might be processed by one model but dropped by another model.

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.

Note:

An event delayed beyond the defined number of seconds is always dropped if any custom input or output blocks written using the version 1 API of the Analytics Builder Block SDK are added as extensions. See the documentation at https://github.com/SoftwareAG/apama-analytics-builder-block-sdk for information on how to migrate such blocks to the version 2 API.

The correlator logs the number of dropped events periodically to the correlator log file. See "Configuration" on page 144 for configuring logging throttling and "Accessing the correlator log" on page 148.

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 126.

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.

Keys for identifying a series of events

Input and output blocks identify a series of events by specifying a key for the series (or stream) of events. This series of events is used to identify correct events to deliver to an input block. The key is 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 is 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 is the device and the event type.
- For `Alarm` objects, the key is the device and the alarm type.

Restrictions for output blocks

In Analytics Builder, for synchronous output types such as measurements, events and alarms (see also "Output blocks and event timing" on page 101), it is not allowed to have more than one output block which generates events with any given key.

As there can be connections between the models, the main reason for this restriction is to avoid ambiguities or errors that may occur while processing events in the input blocks if there are multiple output blocks (in different models) generating the same output stream at the same point in time.

8 Details of Values and Blocks

Introduction

Analytics Builder provides an environment for connecting blocks together to form models that can process and react to inputs. Analytics Builder uses a few types of values internally, and it is important to understand the differences between these. The following topics cover the distinctions between value types representing continuous-time and discrete-time values, and the `pulse` type. They also cover some of the details of block implementations around windowing and when blocks generate output.

Summary

In Analytics Builder, the value types `float`, `boolean`, and `string` are used to represent continuous-time values. They have the following properties, which you should consider when writing models or creating custom blocks:

- A value wire holds its value until there is new input.
- Repeated inputs of the same value should not affect the output (for most cases).
- There is no guarantee that a new input will occur within a defined time period.

Contrast these to the `pulse` type, which represents discrete events and has the following properties:

- A pulse represents a single point in time.
- Multiple inputs of a pulse have significance, even if there is no difference to any value associated with the pulse.
- If a block has multiple input ports for pulses, inputs occurring at different times to different ports will only be "seen" one at a time. An input port for a pulse on a block acts as if it is automatically "reset" after evaluation.

These properties and the rationale behind them are explored and explained in the following topics. These topics also explain how to handle cases that do not fit into these distinctions, such as discrete numeric measurements.

Values as representations of continuous-time physical quantities

A continuous-time value type, especially of the `float` (that is, numeric) type, is typically used to represent the measurement of some continuous physical quantity or property by a sensor. For example, a value may represent one of the following:

- The pressure in a pipe.
- The temperature measured by a thermometer.
- The rotational speed of an axle.
- The position of an object.

These are all continuous measurable properties which are analog in nature. There will be some degree of precision as to how accurately they can be measured in both time and value (and within physical limits). By time accuracy, we mean how frequently a measurement can be made and how precisely the time of the measurement is recorded. There may also be latency - a delay between a change in the actual property and when that can be measured. By value accuracy, we mean with what level of precision the value can be measured - typically at least 2 significant figures, and rarely more than 4 or 5 significant figures of precision can be distinguished. By continuous, we mean that it is valid to measure the property at any point in time.

Taking discrete measurements at different times rather than continuously may be referred to as "sampling" (see also https://en.wikipedia.org/wiki/Sampling_(signal_processing)), and limits as to the value precision may be referred to as "quantization error" (see also https://en.wikipedia.org/wiki/Quantization_(signal_processing)). When measuring a continuous value, the rate at which measurements are obtained should not make significant differences to the output of a block or a model. More measurements may give a more accurate output, but should not make a gross change to calculations.

For example, a sensor measuring the rotational speed of an axle may be able to provide a new measurement every one tenth (0.1) of a second, and only measure in the range 0 to 10000 rpm to the nearest 50 rpm. A change of 10 or 20 rpm may not result in any change of measured value, as the change is less than the level of precision. Applying brakes to a rotating axle to stop it may not be detected immediately, but result in one reading of 1000 rpm, followed by a reading 0.1 seconds later at 0 rpm after the axle has stopped (while the axle would take a few tens of milliseconds to stop, slowing down over that time period).

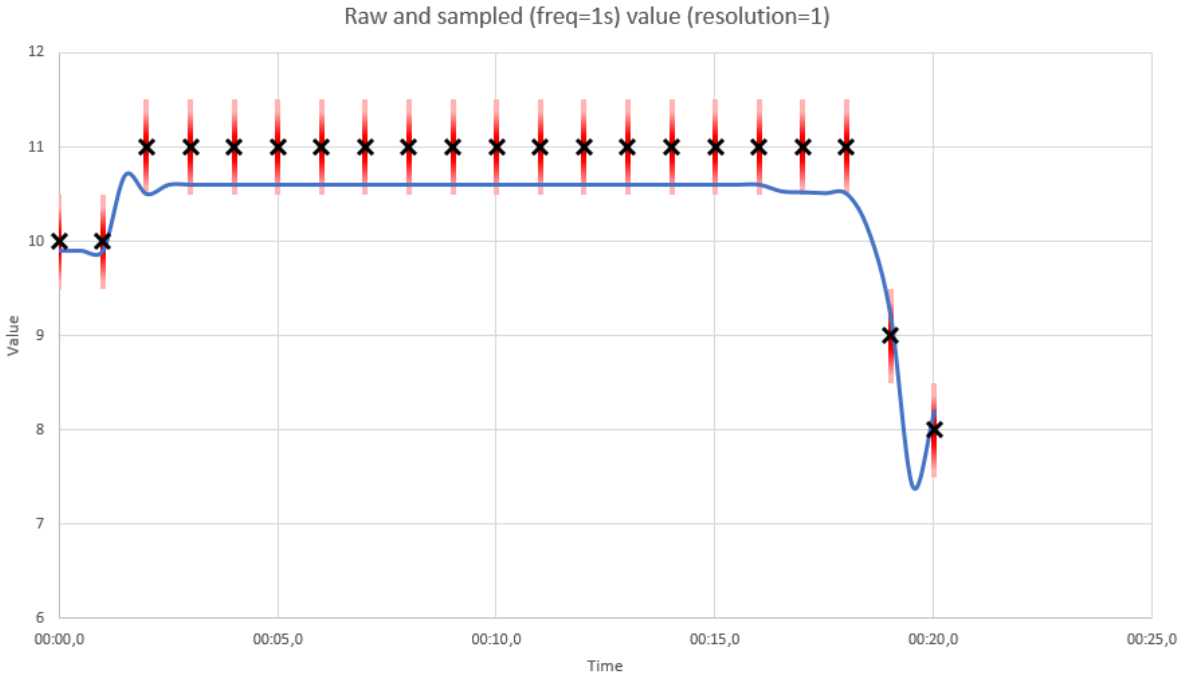
A sensor may be connected in such a way that it provides a new measurement at a regular frequency (for example, audio sampling at 8,000 Hz or a camera taking video at 50 frames a second). This is a *regular sampling input*.

A simple and common optimization is that a sensor or device may generate a new measurement value only if the value is different to the previous value. For many sensors, it would be normal to measure something that often maintains a steady or constant value (at least to within the quantization limit), and there is little value in repeatedly sending the same value. This is an *on-change input*. There will still be an underlying sampling frequency, but new values are only transmitted from the sensor if they are different.

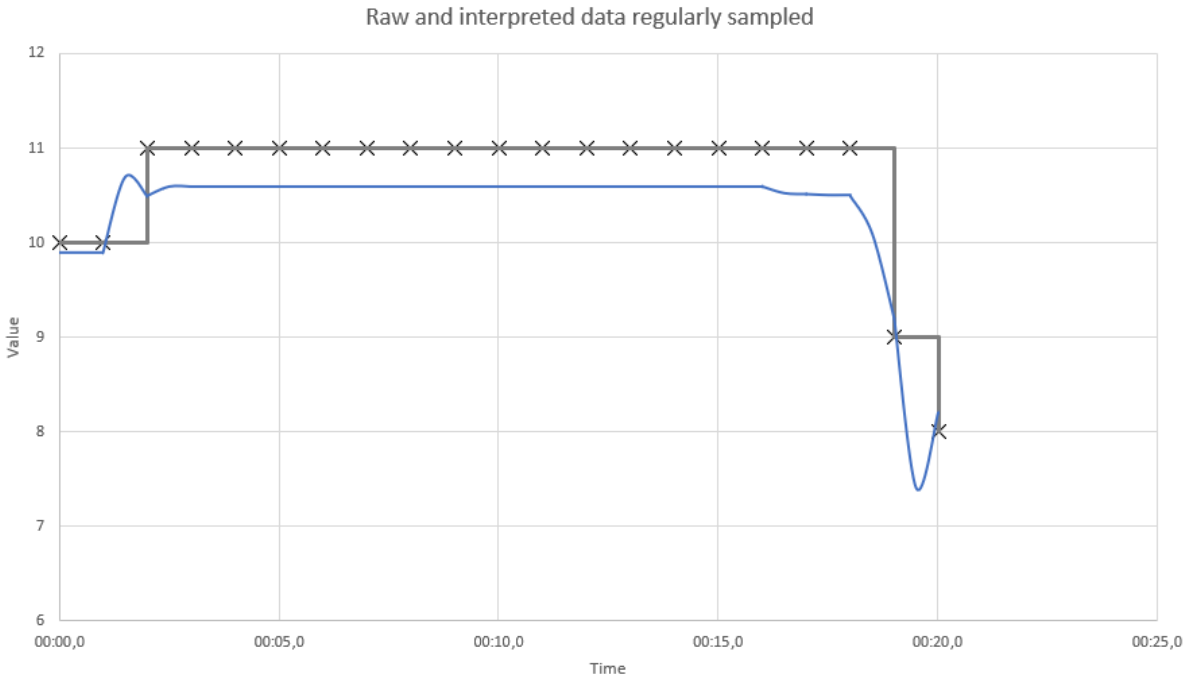
It is also possible to combine the regular sampling and on-change forms together: a sensor that generates a new input if the measurement is different, or periodically. This is a *hybrid input*. For example, the rotational sensor described above may only send a value if the rotation speed changes, or every 10 seconds regardless.

As an example, consider a raw value that changes over time as so:

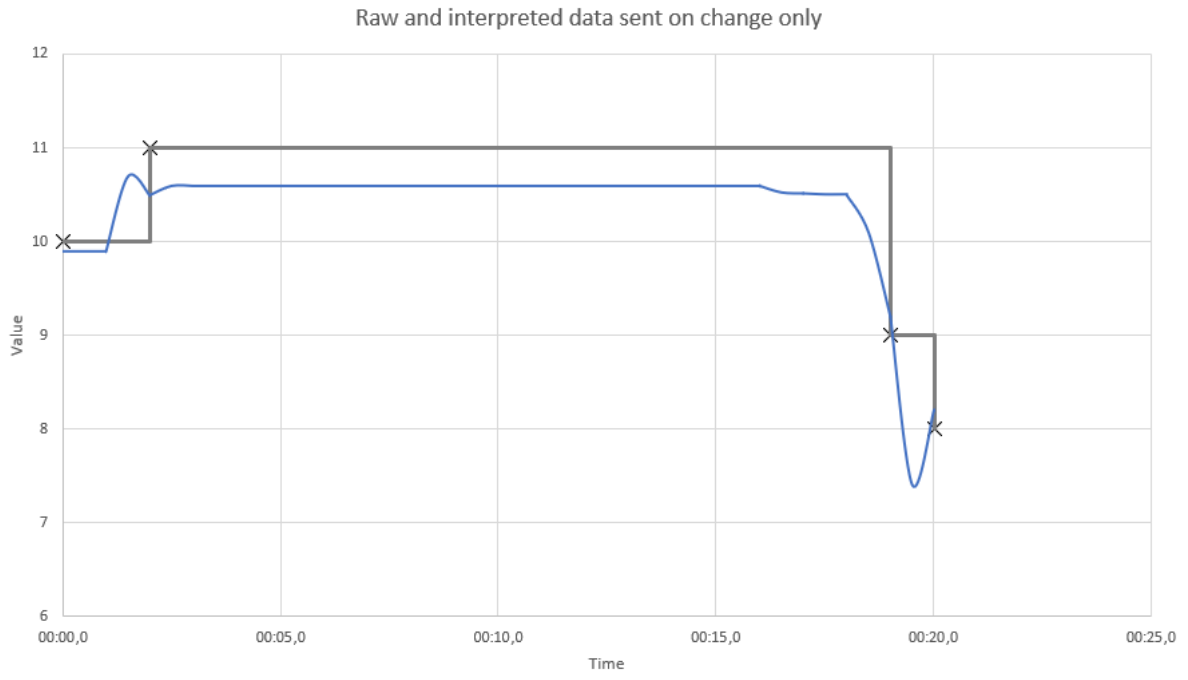
But suppose the sensor can only measure to the nearest whole number, and only once a second. The value thus has some error, shown by the red error bars:



A regular sampling sensor would generate uniform inputs:



While an on-change sensor would generate inputs only when a value changes:



The grey line shows how a real-time processing system such as Apama interprets such values. A value is assumed to maintain the latest value until it is replaced by a newer value. It is also common to draw lines between measurements. So there is a straight line decreasing from value 11 at 00:01 to value 9 at 00:19. However, a real-time system cannot do this. It does not know what the next value is, whereby viewing historic data can interpolate between values. At time 00:19.5, the only information it has is that the value was 11 and then 9. It does not yet know that the value will be

8 at time 00:20. Note that there is no difference between the interpreted grey line in the regularly sampled and on-change-only case. Note that in the middle of the graph, there is a significant quantization error (the `true` value of 10.6 is read as 11), and the sampling frequency of only once a second means that the minimum point of 7.4 at 00:19.5 seconds is lost.

Input values at different times

Consider two position sensors which give the position of two robot arms, and both are *on-change* sensors. If the two arms move together in unison in the same direction and speed, then the position sensors should update to new values at the same time such that they are a constant distance apart (or at least, close to a constant distance). If the **Difference** block has inputs connected to both sensors, then even if the robot arms move, the output of the **Difference** block should be approximately constant. Analytics Builder evaluates all values with the same timestamp, so even though there may be a small delay in receiving the values from the two sensors, provided they supply timestamps from the same clock (and the **Ignore Timestamp** parameter of the **Measurement Input** block is not set), then the **Difference** block will always generate a synchronized output, as shown in the table below:

Time	Position sensor 1	Position sensor 2	Output of the Difference block
00:00	4	14	10
00:01	6	16	10
00:02	9	19	10

By contrast, consider if the two robot arms do not move in unison - one moves, then another. The distance between the arms may vary as either arm moves. As the sensors are *on-change inputs*, there will only be a new value when the position changes. However, the absence of an input does not mean that the corresponding robot arm does not have a position. It has remained where it is (within error margins). For example:

Time	Position sensor 1	Position sensor 2	Output of the Difference block
00:00	9	19	10
00:01	11	**19**	8
00:02	13	**19**	6
00:05	**13**	18	5
00:06	**13**	17	4
00:07	**13**	15	2

The bold numbers indicate the effective value. The last value latches if it has not been replaced by a more up-to-date value.

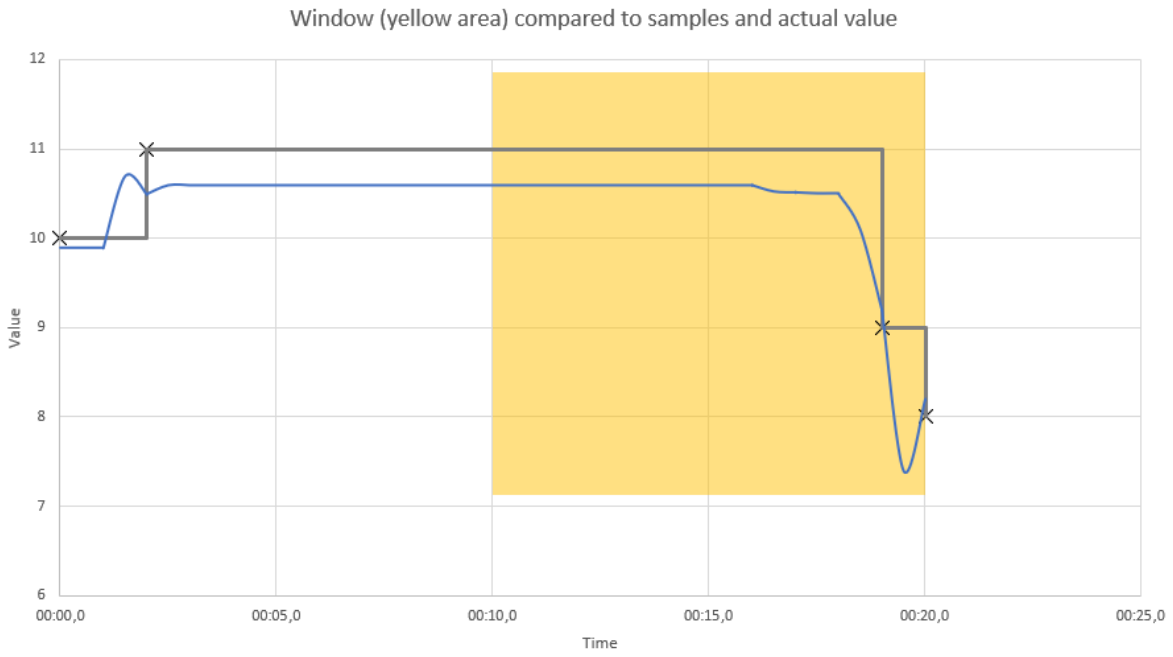
On-change inputs and time windows

If an on-change input is connected to an aggregate block such as the **Average (Mean)** block, then the block should treat the input as continuously having the most recent value it received. This is significant for blocks that maintain a time window. Even if the block last received an input (and thus had its `$process` action called) more than the time window ago, the contents of the window will contain the most recent value. For example, consider the **Average (Mean)** and **Integral** blocks with window duration set to 10 seconds, and input as so:

Time	Input value	Window contents	Output of the Average (Mean) block	Output of the Integral block
00:00	10	0: 10	10	0
00:02	11	0-2: 10	10	20
00:10	**11**	0-2: 10, 2-10: 11	10.8	108
00:12	**11**	2-12: 11	11	110
00:19	9	9-19: 11	11	110
00:20	8	10-19: 11; 19-20: 9	10.8	108

In this case, note how a measurement received at time 00:02 still has influence on the output at 00:19 and later - because it is not replaced until 00:19. Also note that when a new value occurs, it has zero influence on the average or integral - it has not been that value for any time yet. The only exception is for the **Average (Mean)** block when it starts - with an empty window, the output is the input value.

Also refer to the diagram below for what values the window covers at time 00:20:



While only the measurement updates with values 9 and 8 were received within the window, the average value within the window is close to the 11 value. The measurement update for that was received at time 00:02, but as it is a continuous value, it continues to hold the 11 value until time 00:19.

Note that for a block such as **Missing Data**, the absence of input for some time may affect the behavior of the block. If the **Missing Data** block is configured with a 10 second duration, then it would trigger at time 00:12.

If the **Average (Mean)** and **Integral** blocks receive a regular input from a regular sampling sensor, then the block will receive more measurement values, and the comparable table is:

Time	Input value	Window contents	Output of the Average (Mean) block	Output of the Integral block
00:00	**10**	**0: 10**	**10**	**0**
00:01	10	0-1: 10	10	10
00:02	**11**	**0-2: 10**	**10**	**20**
00:03	11	0-2: 10, 2-3: 11	10.333	31
00:04	11	0-2: 10, 2-4: 11	10.5	42
00:05	11	0-2: 10, 2-5: 11	10.6	53
00:06	11	0-2: 10, 2-6: 11	10.667	64
00:07	11	0-2: 10, 2-7: 11	10.714	75

Time	Input value	Window contents	Output of the **Average (Mean)** block	Output of the **Integral** block
00:08	11	0-2: 10, 2-8: 11	10.75	86
00:09	11	0-2: 10, 2-9: 11	10.778	97
00:10	**11**	**0-2: 10, 2-10: 11**	**10.8**	**108**
00.11	11	1-2: 10, 2-11: 11	10.9	109
00:12	**11**	**2-12: 11**	**11**	**110**
00:13	11	3-13: 11	11	110
00:14	11	4-14: 1	11	110
00:15	11	5-15: 11	11	110
00:16	11	6-16: 11	11	110
00:17	11	7-17: 11	11	110
00:18	11	8-18: 11	11	110
00:19	**9**	**9-19: 11**	**11**	**110**
00:20	**8**	**10-19: 11; 19-20: 9**	**10.8**	**108**

Note that the highlighted lines are the same as without the repeated measurements. Repeated measurements of the same value received by these blocks make no difference to what the block would calculate if re-evaluated.

Window block output timings

For aggregate blocks such as the **Average (Mean)** block, the effect of a change of input value means that, if regularly re-evaluated, the output of the block will change, approaching the new value. If there have been different input values received by the block in the past, then a re-evaluation of the block at any point in time is possible, and each may generate a different output.

With the previous example from "On-change inputs and time windows" on page 111, repeatedly re-evaluating an **Average (Mean)** block with a 10 second window will yield the following:

Time	Input value	Output of the **Average (Mean)** block
00:00	10	10
00:01	10	10
00:02	11	10
00:03	11	10.333

Time	Input value	Output of the **Average (Mean)** block
00:04	11	10.5
00:05	11	10.6
00:06	11	10.667
00:07	11	10.714
00:08	11	10.75
00:09	11	10.778
00:10	11	10.8
00.11	11	10.9
00:12	11	11

It would also be possible to re-evaluate the block every half a second, or any fraction of a second - to milliseconds or even smaller times between re-calculations. It is impractical to "continuously" re-evaluate the block (to re-calculate the average value at a given point in time and generate a new output). So when is an appropriate time for the block to evaluate and generate an output?

The **Average (Mean)** block (and others) provide an **Output Threshold** parameter. If this is set, then the block emulates a sensor which generates a new measurement reading if the output changes by the output threshold amount. Thus, if set at 0.1, we get several outputs between 00:02 and 00:03 (when the output is changing from 10 to 10.333), another output between 00:03 and 00:04 (when it reaches 10.4), outputs at 00:04 and 00:05 exactly, another between 00:06 and 00:07 (10.7), then at 00:10 and 00:11 and 00:12. The block calculates at what time the output would vary by more than the output threshold compared to the most recent output, and re-evaluates at that point in time. Thus, the output may occur quite irregularly in time, but output at times such that the values output always differs by an amount equal to the output threshold. The block also re-evaluates on any new inputs, even if there is not a different value to the last input.

As models may wish to perform a calculation with the output of the **Average (Mean)** block at any point in time (for example, to compare to another measurement), a **Sample** input port is also provided, to force a re-evaluation and generate an output value.

Windows and buckets

A number of blocks, primarily those in the **Aggregate** category, maintain a time-based window of input values received in the past. Their output is a calculation based on values within this window. Typically, such blocks offer two distinct ways of managing this window:

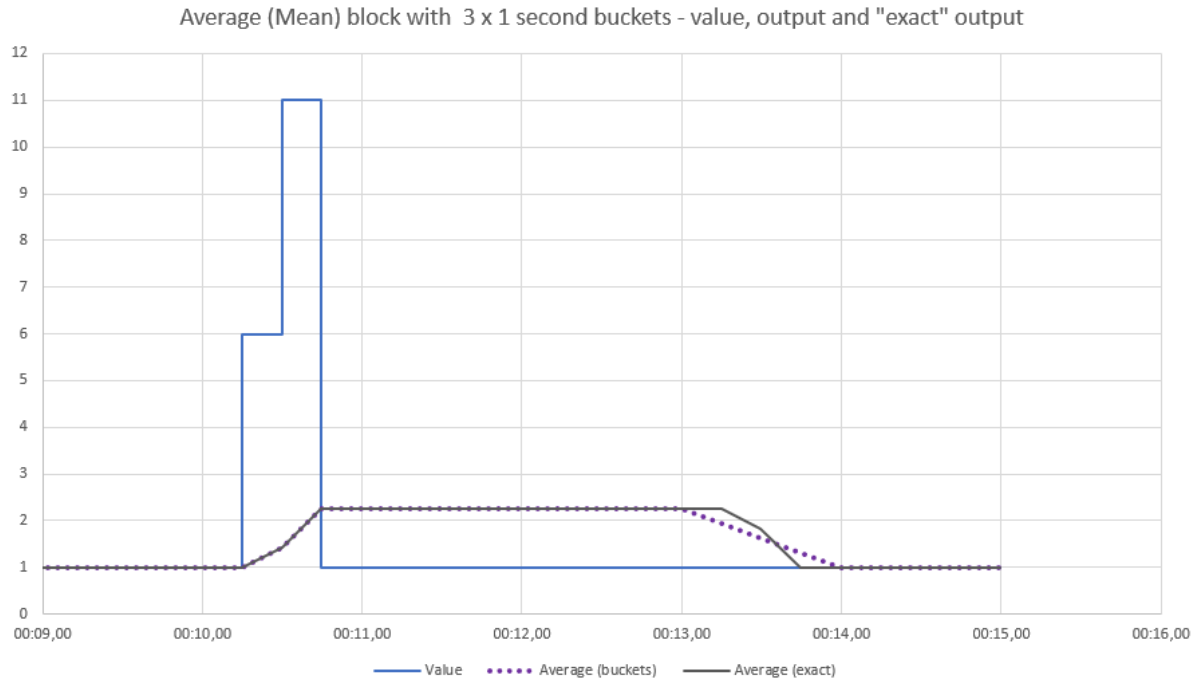
- A parameter value specifying the duration of the window. If set, the window automatically expires interpreted values older than the time specified (where interpreted values are the latest received value at any point in time, as described in "On-change inputs and time windows" on page 111). If the parameter is not specified, then the block does not automatically expire any data.

- A reset input. When a signal is received, the contents of the window is cleared and the block resets to having no contents.

It is possible to use these in combination, or neither, but more typically one or the other.

For the case of the window duration being specified, the block must be able to expire old data. A strictly exact implementation of this would be to store each different measurement input along with the time it occurred. For long windows and/or high frequency inputs, this can result in a large amount of data being stored. To avoid an excessive amount of data being stored, the product blocks do not store all measurement values and times. Instead, the window duration is divided into equal-size buckets. The blocks store state per bucket and use that information to re-calculate the output of the block. A historic bucket can either be completely within the window or be partly expired. If a bucket is partly expired, then the block applies a fractional proportion of the values within that bucket. The practical effect of this is that if the value is changing without significant fluctuations, there is only small difference between an exact (but more resource-intensive) implementation of the block and one that uses buckets. If there is a significant fluctuation in the input value that causes a shift in the output, then the exact time of individual measurement inputs is lost, and the effect that the significant value has as it expires will be spread in time by up to one bucket duration. The product blocks use 20 buckets as a reasonable compromise between accuracy and efficiency.

To illustrate this, we exaggerate the effect by simulating an **Average (Mean)** block with 3 buckets and a 3 second window, so each bucket is 1 second in duration. A few anomalous readings (after a continuous input of value 1) affect the average for both exact and bucket-based **Average (Mean)** blocks in the same way, but we can compare the result of “exactly” expiring each value exactly 3 seconds after it occurred with using buckets, where the change in output is smoothed over the bucket duration:



Note that not only is the timing of the expiry of the anomalous values less precise, the exact shape of the output is lost. The bucketed average changes uniformly between time 00:13 and 00:14. Remember, the product blocks use 20 buckets, so the effect would be less pronounced in this case.

Pulse signals

A pulse is used to signal a point in time or a change of state. Examples of use cases for pulses are:

- A person goes through a gate (for example, at a train station).
- A button is pressed (for example, an emergency stop button).
- A machine goes into a new state (for example, a gateway is reset or powered on).
- A device has made a connection to the network.

In Cumulocity IoT, events, alarms and operations are used as the sources of pulses. See also the following topics in the Cumulocity IoT documentation: Events and Operations in the *Concepts guide* and Alarms in the *Reference guide*.

A pulse may be merely a point in time, but it can also convey extra information, for example, the version number of the software or which network node it has connected to. These can be obtained using the **Extract Property** block. If you are writing your own custom blocks, these are accessible if the input is declared as a `Value` type, which has a properties field. This can be used with numeric value types as well. See the documentation for the Analytics Builder Block SDK for more information on the `Value` type.

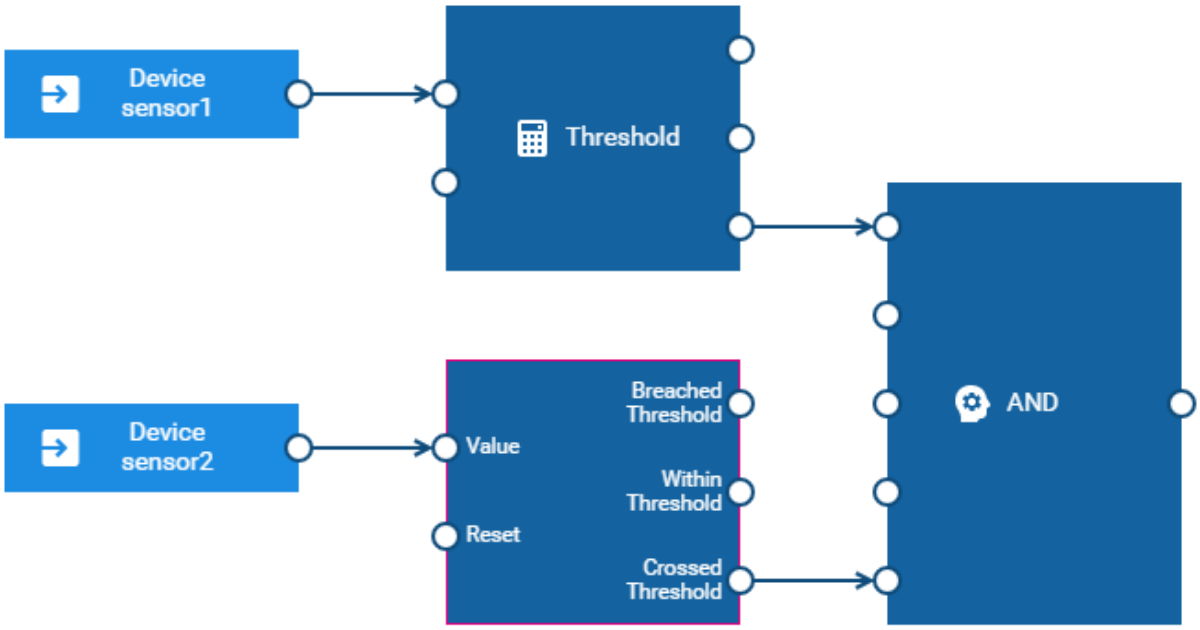
In contrast to measurement values, the timing and number of pulses is very significant, and even though the only difference between subsequent pulses may be the time they were received, each is still significant (whereas multiple measurements with the same value are of little interest).

In contrast to measurement values, a pulse is only active for a single evaluation of a model, where a model evaluation processes all blocks that have a timer that fires (including input blocks) and any blocks connected to outputs that have changed. While both pulses and boolean measurement values are represented by the `boolean` type in the EPL of the blocks, their behavior is different:

- If a boolean measurement value is received by a block, it will "stick" to its value until replaced. For example, if looking at whether temperature sensors 1 and 2 are both over a threshold using an **AND** block, that value is still true after receiving high measurements.
- If a pulse is received by a block, it is reset after it is evaluated. For example, if an **Average (Mean)** block is reset by an event, then the reset happens when the event is received. After that, if no further event is received, the block is not reset on future value inputs.

It is still valid and sensible to combine multiple pulses, for example, with an **AND** block. If two pulses occur at the same point in time, that will be a single evaluation. For example, the **Threshold** block has a **Crossed Threshold** output port which is a pulse that is sent only when a continuous value input goes from one side of the threshold value to another. Two sensors on the same device (thus with the same timestamp) may cross thresholds at the same time, so the **AND** of the output of two such thresholds will only trigger if both inputs cross the threshold with new values of the same timestamp. Note that if one sensor crosses the threshold and then later the other sensor, the

blocks below would never give an output from the **AND** block. They would only do this if the two occurred at the same time.



Discrete-time measurements

There are some cases where a measurement would be used where a numeric measurement value does not represent a continuous-time property. For example:

- The weight of parcels passing a weighing machine on a conveyor belt.
- The size of objects passing a measuring point.
- The value of a ticket scanned or printed by a machine.

In contrast to the continuous-time values, each of these are significant, even if two measurements are of the same value. The time of each measurement may have some significance, but the time between subsequent measurements is of no great significance. If the measurements were received with slightly different timings, or even potentially out of order, this would not signify a difference (for example, the sum of the value of tickets does not change if they are processed in a different order or with different timings, and the time between values is unlikely to be uniform). Note that by discrete-time we are only referring to the *time* of the measurements. The value may still be continuous. For example, weight is a continuous value, but we may weigh individual parcels - while the weight of a parcel may be representable to fractions of a gram of weight. If we are between two parcels on a conveyor belt, there is no “current” value for the weight of the parcel at that point. The value could also be discrete. For example, the ticket value would typically be a discrete value (for example, to the nearest cent, or one of a few predefined ticket values).

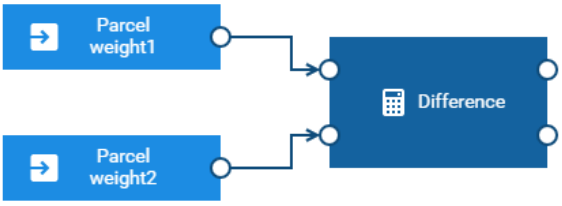
Compare also: https://en.wikipedia.org/wiki/Discrete_time_and_continuous_time and https://en.wikipedia.org/wiki/Continuous_or_discrete_variable. In practice, all measurements are samples of a continuous-time property.

When dealing with discrete-time inputs, you should use the **Discrete Statistics** block rather than the **Average (Mean)** block. While it is possible to connect an input from a parcel weight sensor to the **Average (Mean)** block, the **Average** block weights by time. For example:

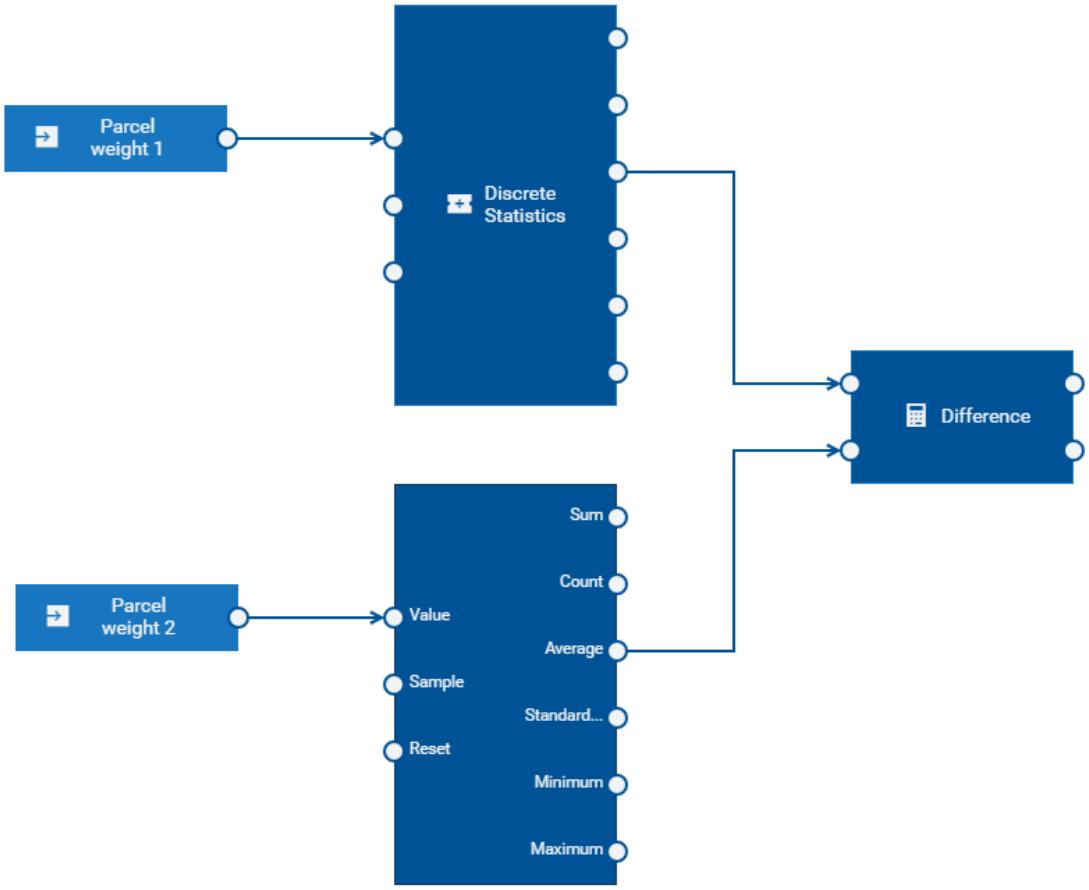
Time	Input value	Average of continuous-time input	Average of discrete-time input
00:10	11	11	11
00:19	9	11	10
00:20	8	10.8	9.33

Compare this to the table in "On-change inputs and time windows" on page 111, looking at times 00:10 onwards (that is, what would be in a window from 00:10 to 00:20). Note that the continuous-time block would generate a different output if the inputs occurred at different times, while a block averaging values based on discrete-time would not.

Note that by default measurements are treated as continuous-time values. So it is possible, for example, to calculate the difference between two values:



The above example gives the difference between the most recent weight received by two sensors. This may not be a particularly useful distinction if these are genuinely discrete-time inputs. However, it can make sense to compare the difference of averages (or means) between two discrete-time inputs. The **Average** output port of the **Discrete Statistics** block gives a continuous-time value:


Parcel weight 1
Discrete Statistics
Difference
Parcel weight 2
Value
Sample
Reset
Sum
Count
Average
Standard...
Minimum
Maximum

9 Models and Devices

Model execution for different devices

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

Each model must either:

- receive input from a set of specific devices and send output to a set of specific devices, or
- receive input from each device within device groups and send output to the trigger device.

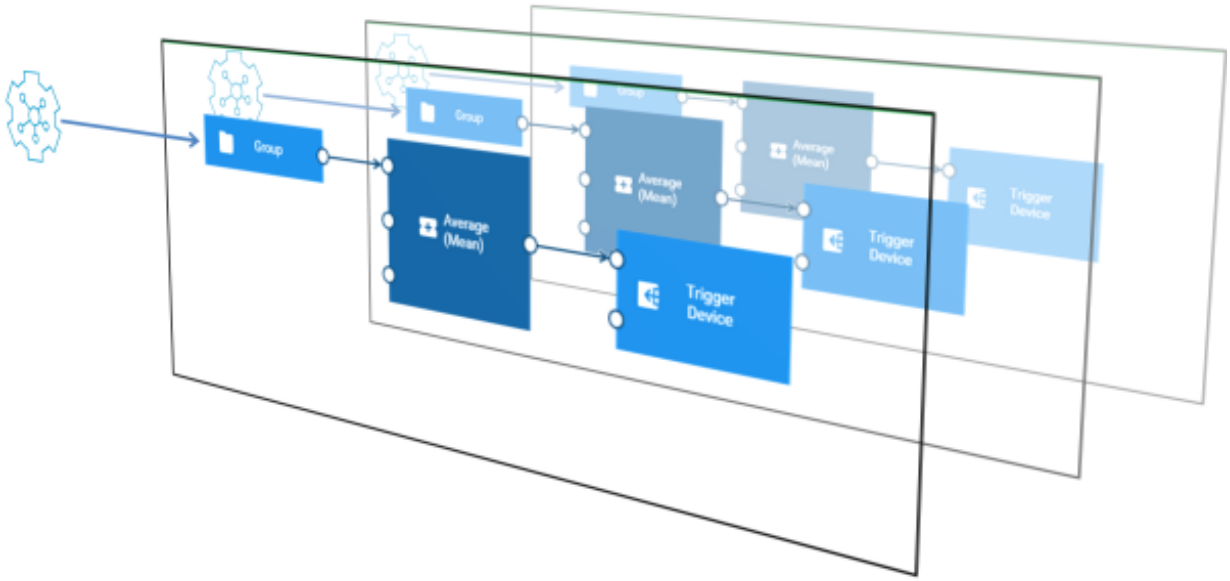
  A device group is a means of organizing devices. A device group can contain any number of devices or other device groups. When a model uses a device group, the model will act on all devices referred to by the device group, either directly or indirectly through members of the device group that are themselves 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 124). However, data from a model processing specific devices can be sent to and received from other models, including models for device groups, and vice versa (see "Connections between models" on page 126).

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 models using specific devices, 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 specific devices, then the output blocks generating outputs can specify the same or different specific devices.

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 instance applies to - or whichever device sent the data to trigger that instance. Asynchronous output blocks in such models can specify the trigger device or any other specific device.

When a template parameter is used for an output block, then if the parameter's value is a device group, then this is treated the same as if it were set to the trigger device. The output will go to whichever device triggered the model's evaluation, with each device within a group being treated independently. Typically, the same template parameter will be used for both input and output, so these will refer to the same group, and each device is processed independently.

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 69.

The test and simulation modes are only permitted for models using specific devices. 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 52 for more information on these modes.

Configuring the concurrency level

By default, the 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 144 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).

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.

Note:

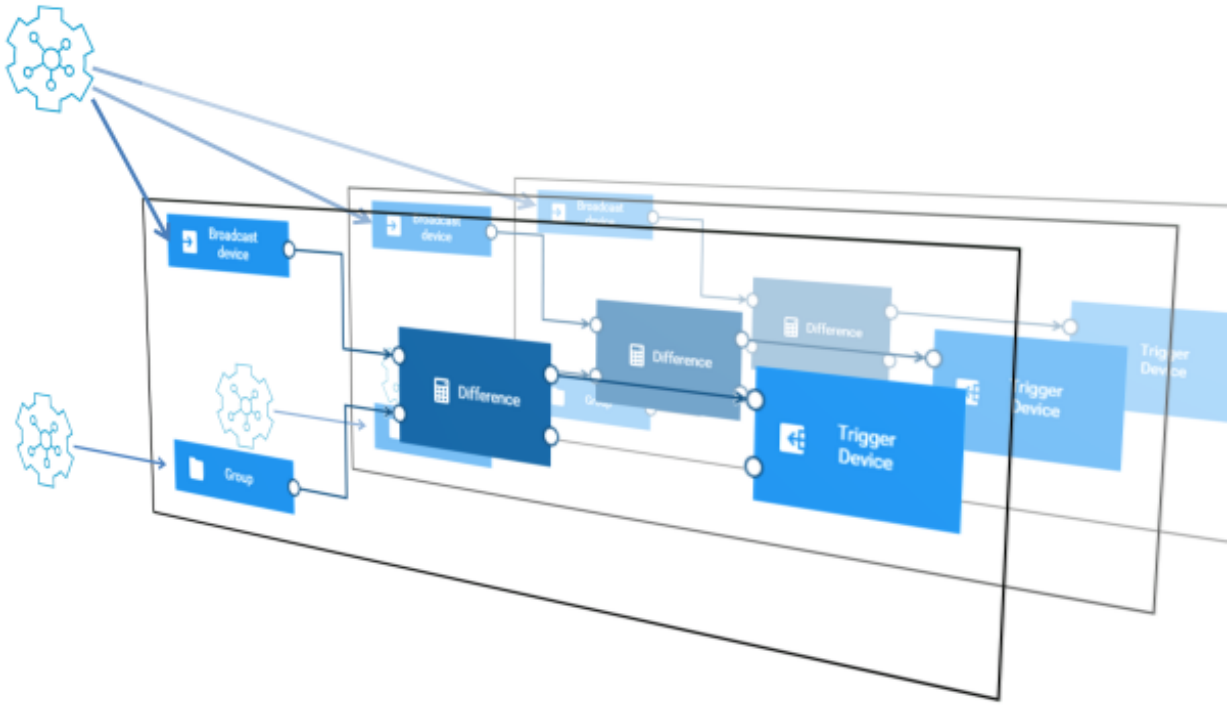
Using multiple specific devices in a model with the concurrency level set to more than 1 can lead to connections between models which are deployed across multiple workers. Chains of models using multiple specific devices with high throughput usually scale less well than chains of models all using a single specific device.

Support for multiple devices in a single model with the concurrency level set to more than 1 is disabled if any input or output blocks written using the version 1 API of the Analytics Builder Block SDK are added as extensions. See the documentation at https://github.com/SoftwareAG/apama-analytics-builder-block-sdk for information on how to migrate the blocks to the version 2 API.

Broadcast devices

It is sometimes useful to have signals that can apply to all models. These may be signals from devices, or from other systems that are presented as if they were signals from a device. 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 specific 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; the presence of either the `pas_broadcastDevice` or `c8y_Kpi` 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. `c8y_Kpi` objects are typically used with the KPI block. Thus, it is possible to use a KPI object to compare measurements from a group of devices - one KPI object is used for all devices in the group.

Virtual devices

A virtual device is used when a model is deployed in test or simulation mode. See also "Deploying a model" on page 52.

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 144.

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. For more details, see "Keys for identifying a series of events" on page 102.

When models are connected together using inputs and outputs for the same stream of events, the term "chain" is used to refer to all of the models that are connected to each other in this way. There may be multiple chains if there are separate groups of models that are connected to each other.

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.

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 97), 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 98), there are restrictions on how output blocks and input blocks can be used to connect models together:

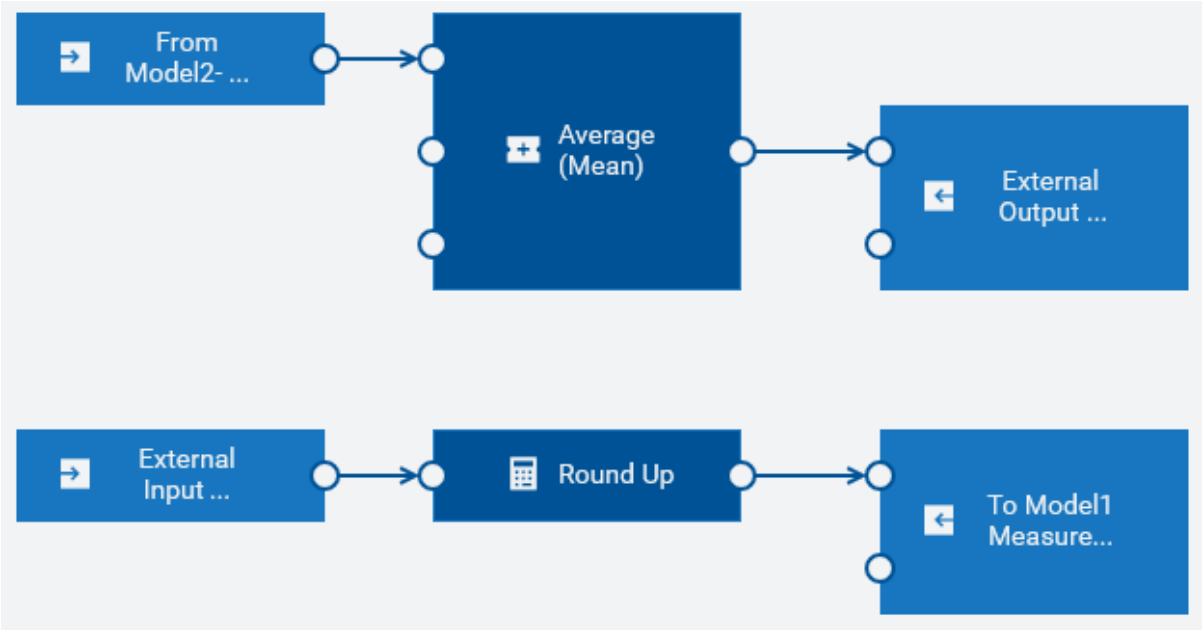
- One block across all models is permitted to generate a series of synchronous events for a given key. See also "Keys for identifying a series of events" on page 102. Multiple output blocks generating asynchronous events can be used within a single model or across multiple models.
- No cycles can be 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).



Note:
Using multiple specific devices in a model with the concurrency level set to more than 1 can lead to connections between models which are deployed across multiple workers. Chains of models using multiple specific devices with high throughput usually scale less well than chains of models all using a single specific device.

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 61). 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 62) and when you replace devices, device groups or assets (see "Replacing devices, device groups and assets" on page 69).

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 144.

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 61). 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 62) and when you replace devices (see "Replacing devices, device groups and assets" on page 69).

See also "Configuration" on page 144.

10 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 52 for more details.

Note:

Simulation mode is only permitted for models using specific devices. 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 125). 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.

Configuring the maximum number of simulation models

By default, a maximum of 3 simulation models can be deployed at a time.

If you want to change this default value (to deploy either more or fewer simulation models at a time), 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 5:

```
{
    "category": "analytics.builder",
    "key": "simulation.maxInstances",
    "value": "5"
}
```

See also "Configuration" on page 144.

Configuring an alternative data source for simulation

By default, the platform database is used to retrieve the historical data for the simulation of an analytic model. You can configure an alternative data source for simulation, for example, if historical data is maintained separately. The data source must support HTTP `GET` operations for the required path and query parameters. The response to the `GET` operation must conform to the standard JSON format of Cumulocity IoT. Refer to the Cumulocity IoT documentation for the data and query parameter formats.

If you want to use an alternative data source for simulation, you need to define the tenant options listed below. That is, you need to send 3 separate `POST /tenant/options` requests. For detailed information, see the information on tenants in the *Reference guide* at https://www.cumulocity.com/guides/.

- `simulation.dataSource.url` - The base HTTP URL of the data source.
- `simulation.dataSource.username` - The username to be used for the HTTP basic authentication.
- `simulation.dataSource.password` - The password to be used for the HTTP basic authentication.

For example:

```
{
    "category": "analytics.builder",
    "key": "simulation.dataSource.url",
    "value": "http://192.168.1.1/"
```

```
}
```

```
{
    "category": "analytics.builder",
    "key": "simulation.dataSource.username",
    "value": "myname"
}
```

```
{
    "category": "analytics.builder",
    "key": "simulation.dataSource.password",
    "value": "secret"
}
```

See also "Configuration" on page 144.

The sections below list the path and query parameters that need to be supported by the alternative data source.

Alarms

Path: `alarm/alarms`

Query parameters:

- `source` - Source ManagedObject identifier of the alarms.
- `dateFrom` - Start date or date and time of the alarm occurrence.
- `dateTo` - End date or date and time of the alarm occurrence.
- `pageSize` - Maximum number of records to return.
- `currentPage` - The current returned page within the full result set, starting at page 1.

Events

Path: `event/events`

Query parameters:

- `source` - Source ManagedObject identifier of the events.
- `dateFrom` - Start date or date and time of the events.
- `dateTo` - End date or date and time of the events.
- `pageSize` - Maximum number of records to return.
- `currentPage` - The current returned page within the full result set, starting at page 1.

Measurements

Path: `measurement/measurements`

Query parameters:

- `source` - Source ManagedObject identifier of the measurements.
- `dateFrom` - Start date or date and time of the measurements.
- `dateTo` - End date or date and time of the measurements.
- `pageSize` - Maximum number of records to return.
- `currentPage` - The current returned page within the full result set, starting at page 1.

Operations

Path: `devicecontrol/operations`

Query parameters:

- `deviceId` - Source device identifier of the operations.
- `createdFrom` - Start creation time of the operations.
- `createdTo` - End creation time of the operations.
- `pageSize` - Maximum number of records to return.
- `currentPage` - The current returned page within the full result set, starting at page 1.

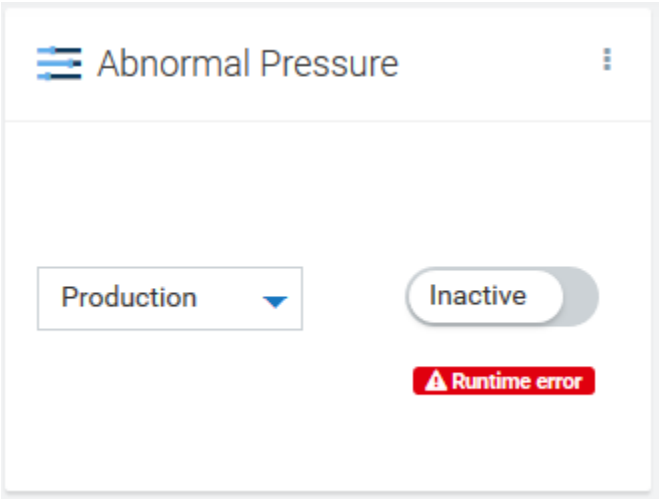
Monitoring dropped inputs

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 142.

11 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, Runtime error 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 55.

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 144 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.
`chain_diagnostics`	Information about model chains. See "Connections between models" on page 126 for more information.
`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.

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*.

Chain diagnostics

The following information is published for all chains that are present:

Name	Description
`creationTime`	The time when this chain was created.
`executionCount`	The number of times the chain was evaluated. [1]

Name	Description
`modelsInEvalOrder`	A list of model identifiers in the order in which the models were evaluated.
`pendingTimersCount`	The number of pending timers which are behind the current time.
`maxTime`	The maximum time taken to evaluate the chain. [1]
`minTime`	The minimum time taken to evaluate the chain. [1]
`meanTime`	The mean time taken to evaluate the chain. [1]
`execBucket`	The execution time statistics of the chain. [1, 2]

- [1] The fields are updated if the chain is evaluated fully or partially. Partial evaluation of a chain means that only some models of the chain are evaluated.
- [2] There are 21 buckets which store the number of times when the execution time falls within the bucket range. Each bucket has size of `timedelay_secs` divided by 10 seconds, except for the last bucket which stretches to infinity. For example, if `timedelay_secs` is 2 seconds, then the first bucket holds the number of times when the chain execution took up to 0.2 seconds, the second bucket holds the number of times when the chain execution took more than 0.2 seconds but up to 0.4 seconds, and so on. See also the following example:

Bucket	Execution time range
1	0 - 0.2
2	0.2 - 0.4
3	0.4 - 0.6
...	...
20	3.8 - 4.0
21	4.0 - infinity

For more information on `timedelay_secs`, see “Keys for model timeouts” on page 145.

Slowest chain status

When chains of models with a high throughput are deployed across multiple workers, it may happen that the chain falls behind in processing input events, creating a backlog of input events that are still to be processed. These chains are referred to as “slow chains”. A message is written to the correlator log if the slowest chain is delayed by more than 1 second. For example:

```
Analytics Builder chain of models "Model 1", "Model 2", "Model 3" is slow by 3 seconds.
```

See “Accessing the correlator log” on page 148 for information on where to find the correlator log.

The following information on the slowest chain is also available in the periodic status that is published as Cumulocity IoT operations or events, within the `apama_status` parameter:

Name	Description
`user-analyticsbuilder.slowestChain.models`	All models contained in the slowest chain.
`user-analyticsbuilder.slowestChain.delaySec`	The number of seconds the chain lags behind in processing the input events.

Example

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


```
{
    "creationTime": "2021-01-05T21:48:54.620+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
        }
    },
    "chain_diagnostics": {
        "780858_780858": {
            "creationTime": 1600252455.164188,
            "executionCount": 4,
            "modelsInEvalOrder": ["780858_780858", "780860_780860"],
            "pendingTimersCount": 1,
            "timeData": {
                "execBucket": [2,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                "maxTime": 0.00014781951904296875,
                "meanTime": 0.0001152356465657552,
                "minTime": 6.29425048828125e-05
            }
        }
    },
    "apama_status": {
       "user-analyticsbuilder.slowestChain.models": "\"Model 1\", \"Model 2\", \"Model
 3\"",
```

```
        "user-analyticsbuilder.slowestChain.delaySec": "3",
        "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"
    }
}
```


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 100.

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 138 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 removed, 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.
```

Deploying a model can combine existing models or chains to form a new chain. The formation of a chain may take a while to complete as it may combine multiple existing models and chains. Activation messages are written to the correlator log whenever the activation of a chain is started and completed. For example:

```
Analytics Builder chain of models "Model 1", "Model 2", "Model 3" is being activated.
Analytics Builder chain of models "Model 1", "Model 2", "Model 3" has been activated.
```

See "Accessing the correlator log" on page 148 for information on where to find the correlator log.

Viewing the audit logs

Model activations and deactivations are shown in the audit logs. The audit logs are accessible via Cumulocity IoT's Administration application and the audit API. See the *User guide* and *Reference guide* at https://www.cumulocity.com/guides/ for details of accessing the audit logs.

Audit log entries include the name of the user performing the action and the current mode of the model. For test and simulation mode, the identifier of the virtual device and the mode properties are provided.

The following is an example of an audit log entry (with additional line breaks in the `text` field for better readability):

```
{
    "activity": "Activated model",
    "application": "apama_ctrl",
    "severity": "information",
    "text": "TestingModel1:Activated,
             Mode:SIMULATION,
             OutputDevices:[21628],
             startTime:2019-11-06T09:35:10.000Z,
             endTime:2019-11-06T09:35:43.000Z,
             maxDelaySecsSimulation:2",
    "type": "Inventory",
    "user": "apama_test_cep_admin"
}
```

Viewing diagnostics information

To view diagnostics information, you need READ permission for "CEP management" in Cumulocity IoT. See the information on the Administration application in the *User guide* at https://www.cumulocity.com/guides/ for details on managing permissions.

Note:
ADMIN permission for "CEP management" do not include READ permission.

If you have READ permission for "CEP management", then links for downloading diagnostics information are available at the bottom of the model manager. These will download zip files that include log file contents, copies of EPL applications, and much more.

It may be useful to capture this diagnostics information when experiencing problems, or for debugging EPL applications. It is also useful to provide to support if you are filing a support ticket.

See the *Streaming Analytics guide* at https://www.cumulocity.com/guides/ for detailed information on the available diagnostics. This also includes information on additional endpoints that are available for REST requests.

Configuration

You can customize the settings of 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 147. However, you can use any tool you like.

To change the tenant options, you need ADMIN permission for "Option management" in Cumulocity IoT. See the information on the Administration application in the *User guide* at https://www.cumulocity.com/guides/ for details on managing permissions.

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`.

Key name	Description
`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 one of the following values: ■ `EVENT` - Publish the status as a Cumulocity IoT event. ■ `MEASUREMENT` - Publish the status as a Cumulocity IoT measurement.
`status_send_keys`	A comma-separated list of field names to be used when publishing the status. If not set or empty, the status for all fields is published. For example, if you specify the following, then the status includes only the values for these fields in one measurement. `numQueuedInput,numListeners,numMonitors`
`status_event_type`	The event type if the status is published as a Cumulocity IoT event, or the measurement type if the status is published as a Cumulocity IoT measurement. 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 100.
`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 100.
`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 98.

Key name	Description
`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.
`block_promise_timeout_secs`	The timeout in seconds to wait for `Promise` values returned by block actions like `$validate` to be completed. A block returns a `Promise` value from an action when it performs an asynchronous operation. If a `Promise` value is not completed within the configured duration, then model deployment fails. The default value is 60 seconds.

Keys for simulation mode

Key name	Description
`simulation.maxInstances`	The maximum number of simulation models to be deployed at a time. The default value is 3 models. See also "Configuring the maximum number of simulation models" on page 133.

`simulation.dataSource.url`
`simulation.dataSource.username`
`simulation.dataSource.password`
	The URL of an alternative data source for simulation. This also requires a user name and password for HTTP basic authentication. See also "Configuring an alternative data source for simulation" on page 133.

Other keys

Key name	Description
`numWorkerThreads`	The number of worker threads. The default value is 1. See also "Configuring the concurrency level" on page 124.
`retention.` `virtualDevicesMaxDays`	The retention period in days for keeping virtual devices. The default value is 30 days. See also "Virtual devices" on page 125.
`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 128.

`c8yAnalyticsBlocks.`
`queryInventoryName`
`SearchAdditionalFilter`
	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 129.

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 148.

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 ADMIN permission for "Option management" 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
  ```

  If your tenant does not have its own unique host name, you have to provide the tenant identifier in the *username* argument. For example:

  ```
  --user management/User123
  ```

  or

  ```
  --user t12345/User123
  ```

- *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.

12 Block Reference

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 (excluding custom blocks that you have created yourself).

Block Name	Description
Alarm Input	Receives `Alarm` objects from a device or device group and reorders them based on the timestamp.
Alarm Output	Creates a new `Alarm` object for a specified device or for the current device with a pre-configured alarm name and parameters.
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.
Constant Value	Outputs a value, either when the Trigger input port receives a signal or at startup.
Counter	Gives a count of the total inputs and repeated inputs.
Cron Timer	Sends a signal output based on cron-like periodic timer syntax.
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.
Discrete Statistics	Generates statistics of sum, count, average (mean), standard deviation, minimum and maximum for discrete input values.
Duration	Measures the time elapsed from a set start time.
Event Input	Receives `Event` objects from a device or device group and reorders them based on the timestamp.
Event Output	Creates a new `Event` object for the associated device or the triggering device.
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.

Block Name	Description
From Base N	Converts a base N string to a float.
Gate	Blocks the input from going to output unless the gate is open and enabled.
Geofence	Compares the input value against the defined geofence value to detect whether the device is within the geofence, and whether the device entered or exited the geofence.
Gradient	Calculates the weighted linear regression gradient for the values.
Group Statistics	Generates periodic aggregate values across all the devices in a group for which the block has received input values.
Integral	Calculates the integral of the input value over time.
KPI	Compares a value against either a KPI (Key Performance Indicator) or the data point of a device.
Latch Values	Latches the latest input value received while the block is enabled.
Limit	Outputs a value that is kept within the defined upper and lower limits.
Machine Learning	Invokes the specified Machine Learning model that scores the input data.
Managed Object Input	Receives `ManagedObject` objects from a device or device group.
Managed Object Output	Updates a `ManagedObject` object for a specified device or for the current device.
Measurement Input	Receives `Measurement` objects from a device or device group and reorders them based on the timestamp.
Measurement Output	Creates a new `Measurement` object for the associated device or the triggering device.
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.
Operation Input	Receives `Operation` objects from a device or device group.
Operation Output	Creates a new `Operation` object for a specified device or for the current device.
OR	Performs a logical 'or' on the inputs.
Position Input	Receives `Event` objects from a device or device group and extracts the `c8y_Position` fragment into a `Value` object.
Pulse	Converts a non-pulse input into a pulse output.

Block Name	Description
Range	Compares the input value against the defined lower and upper range values to detect whether the input is within or out of the range, or whether it crosses the range.
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.
Selector	Outputs a parameter value depending on which input port has a `true` value, lowest number taking precedence.
Send Email	Sends an email to the specified email addresses.
Send SMS	Sends an SMS (Short Message Service) to the specified phone number.
Set Properties	Outputs a pulse with properties set from values on the input ports.
Standard Deviation	Calculates the standard deviation and variance of the values over time.
Switch	Outputs the values from a given input, or acts as a circuit breaker.
Text Substitution	Substitutes identifiers marked with a hash and braces (for example, `#{name}`) in the text template with corresponding entries from the input values.
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.
To Base N	Converts a float to a base N string.
Toggle	Converts two pulse inputs to a boolean output based on the set and reset signals, with optional delays.

Input

Alarm Input

`apama.analyticskit.blocks.cumulocity.AlarmInput`

Receives `Alarm` objects from a device or device group and reorders them based on the timestamp.

If the Alarm Status parameter is Active, then the alarms are reordered based on the timestamp (and dropped if they are too old), unless the Ignore Timestamp parameter is set. Otherwise, data is processed as it is received.

The parameters that define the input stream of the block are "Device or Device Group" and "Alarm Type". If this block is configured with the same "Device or Device Group" and "Alarm Type" parameters as an Alarm Output block in another model, then a connection between the models is formed, as each block refers to the same stream of `Alarm` objects.

Note: When running in simulation mode, because only the creation time of the alarm is stored, the alarm status must be Active.

Parameters

Name	Description	Type	Notes
Alarm Type	The alarm type for which the block will listen.	string	
Device or Device Group	The device or device group from which the alarm has been received. The model editor uses the current device or asset name. This is mapped internally to the inventory identifier.	string	
Severity	The severity of the alarm. If not specified, the block will listen for all alarm severities.	Option - one of: ■ Critical ■ Major ■ Minor ■ Warning	Optional
Alarm Status	The status of the alarm. If not specified, the block will listen for alarms with any status.	Option - one of: ■ Active ■ Acknowledged ■ Cleared	Optional
Notification Mode	Filters `Alarm` events such that only new alarms, updated alarms, or all alarms are processed. The default is that all alarms are processed.	Option - one of: ■ All ■ Updates only ■ New alarms only	Default: All
Ignore Timestamp	If selected, the timestamp of the incoming alarm is ignored. Note: When running in simulation mode, because historical input data is used, timestamps are not ignored.	boolean	Default: false

Output Port Details

Name	Description	Type
Alarms	Generates a pulse output for each `Alarm` object received, with extra properties.	pulse

Event Input

`apama.analyticskit.blocks.cumulocity.DeviceEventInput`

Receives `Event` objects from a device or device group and reorders them based on the timestamp.

If the Ignore Timestamp parameter is set, the block ignores the timestamp of the event and processes the events as they are received, otherwise it drops old events.

The parameters that define the input stream of the block are "Device or Device Group" and "Event Type". If this block is configured with the same "Device or Device Group" and "Event Type" parameters as an Event Output block in another model, then a connection between the models is formed, as each block refers to the same stream of `Event` objects.

Note: When running in simulation mode, because historical input data is used, timestamps are not ignored.

Note: A history of changes is not maintained for `Event` objects, and it is thus not possible to retrieve the original objects from the inventory. For this reason, a model which contains this input block type may behave differently in simulation mode than it would in production mode.

Parameters

Name	Description	Type	Notes
Device or Device Group	The device or device group from which the event has been received. The model editor uses the current device or asset name. This is mapped internally to the inventory identifier.	string	
Event Type	The event type for which the block will listen.	string	
Notification Mode	Filters `Event` events such that only new events, updated events, or all events are processed. The default is that all events are processed.	Option - one of: ■ All ■ Updates only ■ New events only	Default: All

Name	Description	Type	Notes
Ignore Timestamp	If selected, the timestamp of the incoming event is ignored. Note: When running in simulation mode, because historical input data is used, timestamps are not ignored.	boolean	Default: false

Output Port Details

Name	Description	Type
Events	Generates a pulse output for each `Event` object received, with extra properties.	pulse

Managed Object Input

`apama.analyticskit.blocks.cumulocity.ManagedObjectInput`

Receives `ManagedObject` objects from a device or device group.

The block does not reorder the received `ManagedObject` objects and processes them as they are received. If the Property Name parameter is supplied, then the block does not produce new output if the value of the specified property has not changed since the last output, even if other properties on the same `ManagedObject` object have changed.

The Value output from the block contains all properties on the `ManagedObject` object, including the property specified by the Property Name parameter. Property values can be accessed using the Extract Property block.

Properties with values of type `string`, `boolean` or `float` can be accessed by specifying the name of the property in the Extract Property block. For example, if the name of the property is `ap_State`, then specify `ap_State` as the value for the Property Path parameter of the Extract Property block.

If a property value is of type JSON object or sequence, then nested values can be accessed by specifying the full path to the nested values as the name of the property.

For example, if the name of the property is `c8y_SpeedMeasurement` and the value is `{ "Speed": { "value": 1234, "unit": "km/h" } }` (in JSON form), then specify `c8y_SpeedMeasurement.Speed.unit` as the value for the Property Path parameter of the Extract Property block to extract the value of the unit.

Any position data associated with the `ManagedObject` object is available as a `c8y_Position` property and can be extracted using the Extract Property block.

If the value of the property specified by the Property Name parameter of this block is of type `string`, `boolean` or `float`, then the value is also directly available in the Value output port and can be directly consumed by blocks consuming values of that type without using the Extract Property block, for example, the Expression or Difference blocks.

The parameters that define the input stream of the block are "Device or Device Group" and "Property Name".

Parameters

Name	Description	Type	Notes
Device or Device Group	The device or device group from which the managed object has been received. The model editor uses the current device or asset name. This is mapped internally to the inventory identifier.	string	
Property Name	The name of the property for which to listen. The `ManagedObject` object must have a property of this name otherwise, it will be ignored. If not set, the objects are not filtered - every update will generate a pulse output with all of the properties from the `ManagedObject`.	string	Optional
Capture Start Value	Outputs the initial value when the model is activated.	boolean	Default: false

Output Port Details

Name	Description	Type
Value	Generates an output for each `ManagedObject` object received. All properties of the managed object are available as extra properties. You can use the Extract Property block to access their values.	any

Measurement Input

`apama.analyticskit.blocks.cumulocity.DeviceMeasurementInput`

Receives `Measurement` objects from a device or device group and reorders them based on the timestamp.

If the Ignore Timestamp parameter is set, the block ignores the timestamp of the measurement and processes the measurements as they are received, otherwise it drops old measurements.

If using a group for input, select a device within the group to select the fragment and series, and then change to the desired group.

The parameters that define the output stream of the block are "Device or Device Group" and "Fragment and Series". If this block is configured with the same "Device or Device Group" and "Fragment and Series" parameters as a Measurement Output block in another model, then a connection between the models is formed, as each block refers to the same stream of `Measurement` objects.

Note: When running in simulation mode, because historical input data is used, timestamps are not ignored.

Parameters

Name	Description	Type	Notes
Device or Device Group	The device or device group from which the measurement has been received. The model editor uses the current device or asset name. This is mapped internally to the inventory identifier.	string	
Fragment and Series	The fragment for which the block will listen. This only shows fragments and series for measurements associated with the object (device or group) selected. Any measurements on a device within a group will only be shown when a device is selected (unless there are measurements with the group as the source).	string	
Ignore Timestamp	If selected, the timestamp of the incoming measurement is ignored. Note: When running in simulation mode, because historical input data is used, timestamps are not ignored.	boolean	Default: false

Output Port Details

Name	Description	Type
Value	The numeric value from the measurement object.	float

Operation Input

`apama.analyticskit.blocks.cumulocity.OperationInput`

Receives `Operation` objects from a device or device group.

The block does not reorder the received `Operation` objects and processes the operations as they are received. The block can be optionally configured to only process operations having a specified status or property.

The output from the block contains all properties on the `Operation` object. Property values can be accessed using the Extract Property block.

Properties with values of type `string`, `boolean` or `float` can be accessed by specifying the name of the property in the Extract Property block. For example, if the name of the property is `ap_State`, then specify `ap_State` as the value for the Property Path parameter of the Extract Property block.

If a property value is of type JSON object or sequence, then nested values can be accessed by specifying the full path to the nested values as the name of the property.

For example, if the name of the property is `c8y_SpeedMeasurement` and the value is `{ "Speed": { "value": 1234, "unit": "km/h" } }` (in JSON form), then specify `c8y_SpeedMeasurement.Speed.unit` as the value for the Property Path parameter of the Extract Property block to extract the value of the unit.

The parameter that defines the input stream of the block is Device or Device Group.

Note: A history of changes is not maintained for `Operation` objects, and it is thus not possible to retrieve the original objects from the inventory. For this reason, a model which contains this input block type may behave differently in simulation mode than it would in production mode.

Parameters

Name	Description	Type	Notes
Device or Device Group	The device or device group from which the operation has been received. The model editor uses the current device or asset name. This is mapped internally to the inventory identifier.	string	
Operation Name	The name of the operation for which the block will listen. If specified, the `Operation` object must have a property of this name otherwise, it will be ignored.	string	Optional
Operation Status	The status for which to listen. If not specified, the block will listen for operations with any status.	Option - one of: ■ SUCCESSFUL ■ FAILED ■ EXECUTING ■ PENDING	Optional
Notification Mode	Filters `Operation` events such that only new operations, updated operations, or all operations are processed. The default is that all operations are processed.	Option - one of: ■ All	Default: All

Name	Description	Type	Notes
		■ Updates only ■ New operations only	

Output Port Details

Name	Description	Type
Operations	Generates a pulse output for each `Operation` object received. All properties of the `Operation` object are available as extra properties. You can use the Extract Property block to access their values.	pulse

Position Input

`apama.analyticsbuilder.blocks.PositionInput`

Receives `Event` objects from a device or device group and extracts the `c8y_Position` fragment into a `Value` object.

If no `c8y_Position` fragment is present, the event is ignored. If the fragment does not contain at least a valid latitude and valid longitude, the event is ignored. If the Primary Value parameter is set to Altitude and the fragment does not contain an altitude, the event is ignored. Latitudes must be between -90 and 90 degrees inclusive. Longitudes must be between -180 and 180 degrees inclusive.

The primary value of the output `Value` object can be set to be the latitude, longitude or altitude. All members of the `c8y_Position` fragment are added to the properties dictionary of the `Value` object.

If the Ignore Timestamp parameter is set, the block ignores the timestamp of the event and processes the measurements as they are received, otherwise it reorders the events and drops old measurements.

The parameter that defines the input stream of the block is Device or Device Group, and Event Type if set.

Note: When running in simulation mode, because historical input data is used, timestamps are not ignored.

Note: A history of changes is not maintained for `Event` objects, and it is thus not possible to retrieve the original objects from the inventory. For this reason, a model which contains this input block type may behave differently in simulation mode than it would in production mode.

Parameters

Name	Description	Type	Notes
Device or Device Group	The device or device group from which the position is received. The model editor uses the current device or asset name. This is mapped internally to the inventory identifier.	string	
Event Type	The event type for which the block listens. If left unset, then there is no filtering by type. To consume events from another model, this property must be set.	string	Optional
Notification Mode	Filters `Event` events such that only new events, updated events, or all events are processed. The default is that all events are processed.	Option - one of: ■ All ■ Updates only ■ New events only	Default: New events only
Ignore Timestamp	If selected, the timestamp of the incoming measurement is ignored. Note: When running in simulation mode, because historical input data is used, timestamps are not ignored.	boolean	Default: false
Primary Value	The primary value to be output by the block: latitude, longitude or altitude, or empty if not set.	Option - one of: ■ Latitude ■ Longitude ■ Altitude	Optional

Output Port Details

Name	Description	Type
Position	An object containing at least latitude and longitude.	any

Output

Alarm Output

`apama.analyticskit.blocks.cumulocity.CreateAlarm`

Creates a new `Alarm` object for a specified device or for the current device with a pre-configured alarm name and parameters.

This block produces synchronous output. The parameters that define the output stream are "Device or Trigger Device" and "Alarm Type". Using another instance of this block with the same output stream is not allowed.

If "Create Alarm" is provided repeatedly, then the count of the alarm is increased (unless it has been cleared). If the severity has changed, then the count is not increased - the existing alarm is modified instead.

Parameters

Name	Description	Type	Notes
Device or Trigger Device	The device (or for models handling groups, Trigger Device) that the alarm is associated with. The model editor uses the device name. This is mapped internally to the device identifier.	any	
Alarm Type	Identifies the type of this alarm, for example "com_cumulocity_events_TamperEvent".	string	
Message	The text that is displayed if the alarm is triggered. The message is set either by parameter or input. If both are not set, the model name is used. You cannot set both.	string	Optional
Severity	The severity of the alarm. The severity is set either by this parameter or the input port. You must set only one of them.	Option - one of: ■ Critical ■ Major ■ Minor ■ Warning	Optional
Params Fragment	If this parameter is set, then the incoming properties on the Properties input port are added to this fragment in the generated `Alarm`. If this is not set, the properties are copied to the top level of the `Alarm` params.	string	Optional

Name	Description	Type	Notes
Create Asynchronous Output	Allow Create Alarm or Clear Alarm to be done asynchronously. Asynchronous output events are events that do not have source timestamps and can only be consumed by another model in a time-asynchronous manner when they are received back from the platform.	boolean	Optional

Input Port Details

Name	Description	Type
Create Alarm	Creates an alarm when a signal is received.	pulse
Clear Alarm	Sets the alarm status to CLEARED if true.	boolean
Severity	The severity of the alarm - should be one of the severity values (WARNING, MINOR, MAJOR, CRITICAL) or CLEARED.	string
Message	The message for the alarm.	string
Time	Sets the timestamp of the alarm. If not connected, the current model time is used.	float
Properties	The properties to set on the alarm.	any

Event Output

`apama.analyticskit.blocks.cumulocity.CreateEvent`

Creates a new `Event` object for the associated device or the triggering device.

This block sends a new event to the current device or the device specified. The text of the event is determined by the Text Input input port or by the Message parameter. You must not set both. If neither are set, the model name is used as the text.

This block produces synchronous output. The parameters that define the output stream of the block are "Device or Trigger Device" and "Event Type". Using another instance of this block with the same output stream is not allowed.

Parameters

Name	Description	Type	Notes
Device or Trigger Device	The device (or for models handling groups, Trigger Device) to which the event is to be sent. The model editor uses the current device name. This is mapped internally to the device identifier.	any	
Event Type	Identifies the type of this event.	string	
Message	The text that will be displayed when the event is created. This requires that the Text Input input is not connected. If neither are set, the model name is used as the text.	string	Optional
Params Fragment	If this parameter is set, then the incoming properties on the Properties input port are added to this fragment in the generated `Event`. If this is not set, the properties are copied to the top level of the `Event` params.	string	Optional

Input Port Details

Name	Description	Type
Create Event	Creates an event when a signal is received.	pulse
Text Input	Sets the text of the event. The Message parameter must not be set if this is used.	string
Time	Sets the timestamp of the event. If not connected, the current model time is used.	float
Properties	The properties to set on the event.	any

Managed Object Output

`apama.analyticskit.blocks.cumulocity.ManagedObjectOutput`

Updates a `ManagedObject` object for a specified device or for the current device.

If the Property Name parameter is set, then the Value input port (or its properties if it has an empty value) is used to set that property on the managed object.

If the Property Name parameter is not set, then all properties from the Value input port are used to update the managed object.

Note: The following reference properties on a managed object cannot be updated using this block: `childDeviceIds`, `childAssetIds`, `deviceParentIds` and `assetParentIds`.

This block does not participate in time-synchronous model-to-model communication. Multiple blocks can be used in a single model or multiple models to update the same property of the same device. Cycles among models can be formed because of this block, so care must be taken not to introduce indefinite cyclic execution of models.

This block produces asynchronous output. The parameter that defines the output stream of the block is Device or Device Group.

Parameters

Name	Description	Type	Notes
Device or Trigger Device	The device (or for models handling groups, Trigger Device) that the managed object is associated with. The model editor uses the current device name. This is mapped internally to the device identifier.	any	
Property Name	The name of the property to update. If not set, then the properties from the Value input port are mapped to the top level properties of the managed object.	string	Optional

Input Port Details

Name	Description	Type
Value	The value of the property to set.	any
Update Property	Signals that the property is to be updated. If not connected, every new value from the Value input port updates the property.	pulse

Measurement Output

`apama.analyticskit.blocks.cumulocity.CreateMeasurement`

Creates a new `Measurement` object for the associated device or the triggering device.

This block sends a new measurement to a device or device group for the Value input port. If the Send input port is connected, this block only sends an output on receiving a send signal. The measurement is sent to the current device or the device specified.

This block produces synchronous output. The parameters that define the output stream of the block are "Device or Device Group", "Fragment Name" and "Series Name". Using another instance of this block with the same output stream is not allowed.

Non-finite values are ignored.

Parameters

Name	Description	Type	Notes
Device or Trigger Device	The device (or for models handling groups, Trigger Device) to which the measurement is to be sent. The model editor uses the current device name. This is mapped internally to the device identifier.	any	
Fragment Name	The name of the fragment in the measurement.	string	
Series Name	The name of the series in the measurement.	string	
Unit	The name of the unit of measurement (for example, "mm" or "lux").	string	Optional
Params Fragment	If this parameter is set, then the incoming properties on the Properties input port are added to this fragment in the generated `Measurement`. If this is not set, the properties are copied to the top level of the `Measurement` params.	string	Optional

Input Port Details

Name	Description	Type
Value	The measurement to be sent.	float
Send	Signals that a new measurement is to be created. If not connected, every new Value input creates a measurement.	pulse
Time	Sets the timestamp of the measurement. If not connected, the current model time is used.	float
Properties	The properties to set on the measurement.	any

Operation Output

`apama.analyticskit.blocks.cumulocity.CreateOperationStaticValue`

Creates a new `Operation` object for a specified device or for the current device.

If none of the Operation Name, Parameter Name or Parameter Value parameters are set, then the operation will use the Properties input port to populate the `Operation` object.

The block does not participate in time-synchronous model-to-model communication. Multiple blocks can be used in a single model or multiple models to create new operations for the same device. Cycles among models can be formed because of this block, so care must be taken not to introduce indefinite cyclic execution of models.

The block produces asynchronous output. The parameter that defines the output stream of the block is Device or Trigger Device.

Parameters

Name	Description	Type	Notes
Device or Trigger Device	The device (or for models handling groups, Trigger Device) to which the operation is to be sent. The model editor uses the device name. This is mapped internally to the device identifier.	any	
Operation Name	The name of the property to create on the `Operation` object.	string	Optional
Parameter Name	The name of the parameter for the operation.	string	Optional
Parameter Value	The value of the parameter for the operation.	string	Optional
Description	The description of the operation to create.	string	Optional

Input Port Details

Name	Description	Type
Create Operation	Creates an operation when a signal is received.	pulse
Properties	The properties to set on the operation.	any

Send Email

`apama.analyticskit.blocks.cumulocity.Send_Email`

Sends an email to the specified email addresses.

The subject and text must each be provided using either a parameter or via an input port, not both.

Note: When running in simulation or test mode, the block logs the output instead of sending an email.

Parameters

Name	Description	Type	Notes
Subject	The subject of the email.	string	Optional
Text	The text of the email.	string	Optional
Reply to	The reply-to address for the email.	string	Optional
To	The recipients of the email. One or more email addresses separated by commas.	string	Optional
CC	(Carbon copy) The recipients that are to receive a copy of the email. One or more email addresses separated by commas.	string	Optional
BCC	(Blind carbon copy) The recipients that are to receive a blind copy of the email. One or more email addresses separated by commas. Recipients listed in To and CC will not be able to see the BCC list.	string	Optional

Input Port Details

Name	Description	Type
Send	Sends an email when a signal is received.	pulse
Subject	Subject of the email.	string
Text	Text of the email.	string

Send SMS

`apama.analyticskit.blocks.cumulocity.Send_SMS`

Sends an SMS (Short Message Service) to the specified phone number.

If the Send SMS input port is connected, this block sends an output to the specified phone number when receiving a signal.

Note: When running in simulation or test mode, the block logs the output instead of sending an SMS.

Parameters

Name	Description	Type	Notes
Phone Number	The phone number to which the SMS is to be sent.	string	
Text	The content of the SMS. The maximum length is 160 characters. Do not specify this parameter if using the SMS Text input port.	string	Optional

Input Port Details

Name	Description	Type
Send SMS	Triggers sending an SMS.	pulse
SMS Text	Content of the SMS. If not connected, specify the content in the Text parameter.	string

Logic

AND

`apama.analyticskit.blocks.core.And`

Performs a logical 'and' on the inputs.

If all the connected inputs are `true`, then the output will be `true`, else `false`. If any of the inputs is of type `pulse`, then the output will be a `pulse`. Any combination of `pulse` and `boolean` inputs is valid. The block functions even if some inputs are not connected.

Input Port Details

Name	Description	Type
Value 1	First input value to the block.	boolean
Value 2	Second input value to the block.	boolean
Value 3	Third input value to the block.	boolean
Value 4	Fourth input value to the block.	boolean
Value 5	Fifth input value to the block.	boolean

Output Port Details

Name	Description	Type
AND	The logical 'and' of the inputs.	pulseOrBoolean(value1, value2, value3, value4, value5)

NOT

`apama.analyticskit.blocks.core.Not`

Performs a logical 'not' on the input.

If the connected input is `true`, then the output will be `false`. If the connected input is `false`, then the output will be `true`.

Input Port Details

Name	Description	Type
Value	Input value to the block.	boolean

Output Port Details

Name	Description	Type
NOT	The logical 'not' of the input.	boolean

OR

`apama.analyticskit.blocks.core.Or`

Performs a logical 'or' on the inputs.

If any of the connected inputs is `true`, then the output will be `true`, else `false`. If all the inputs are of type `pulse`, then the output will be a `pulse`. Inputs must be of the same type. The block functions even if some inputs are not connected.

Input Port Details

Name	Description	Type
Value 1	First input value to the block.	boolean

Name	Description	Type
Value 2	Second input value to the block.	boolean
Value 3	Third input value to the block.	boolean
Value 4	Fourth input value to the block.	boolean
Value 5	Fifth input value to the block.	boolean

Output Port Details

Name	Description	Type
OR	The logical 'or' of the inputs.	sameAsAll(value1, value2, value3, value4, value5)

Calculation

Crossing Counter

`apama.analyticskit.blocks.core.CrossingCounter`

Detects and counts the number of threshold crossings in the specified direction.

Crossing is defined as a change in the input value from one side of the threshold to the other side of the threshold (that is, from less than to greater than or vice versa).

The block can operate over a time-bounded window that is specified with the Window Duration parameter. If this parameter is not specified, it uses an unbounded window. The Reset input clears the window contents.

If a window is configured, the block will use a set of 20 buckets, so the time of expired values is an approximation to the nearest bucket interval.

Parameters

Name	Description	Type	Notes
Threshold Value	This value is compared against the input value.	float	Default: 0.5
Direction	The direction in which to check for a threshold crossing: whether to detect a crossing in the upwards direction, in the downwards direction, or in both directions.	Option - one of: ■ Upwards ■ Downwards	

Name	Description	Type	Notes
		■ Both	
Window Duration (secs)	If present, the amount of time (in seconds) for which values are to be kept in the window. This must be a finite and positive number.	float	Optional

Input Port Details

Name	Description	Type
Value	The input value for which to detect a crossing.	float
Reset	Resets the count of crossings.	pulse
Sample	Forces re-evaluation of the current value and sends the output.	boolean

Output Port Details

Name	Description	Type
Crossing Count	The number of crossings.	float
Crossing	Sends a pulse when a crossing is detected.	pulse

Delta

`apama.analyticskit.blocks.core.Delta`

Calculates the difference between successive input values.

The block generates an output after getting at least two input values.

Input Port Details

Name	Description	Type
Value	Value to calculate the delta.	float
Reset	Resets the state of the block.	pulse

Output Port Details

Name	Description	Type
Delta Value	The delta value.	float

Difference

`apama.analyticskit.blocks.core.Difference`

Calculates the absolute and signed differences between the connected inputs.

Only generates an output if both inputs receive a value.

Input Port Details

Name	Description	Type
Value 1	First input to the block.	float
Value 2	Second input to the block.	float

Output Port Details

Name	Description	Type
Absolute Difference	The absolute difference of the inputs.	float
Signed Difference	The signed difference of the inputs. Positive if the Value 1 input is larger than the Value 2 input.	float

Direction Detection

`apama.analyticskit.blocks.core.DirectionDetector`

Detects whether the input value changes direction.

Outputs the change in direction and the last significant inflection point, ignoring minor variations if the changes are less than the defined hysteresis value. Repeated inputs of the same value are ignored.

Parameters

Name	Description	Type	Notes
Hysteresis	Only counts a change in direction if the input value changes by the defined hysteresis value since the point of changing direction. Must either be zero (meaning all changes of direction are counted) or a a positive number.	float	Default: 0.0

Input Port Details

Name	Description	Type
Value	Numeric value for which to detect the change in direction.	float
Reset	Resets the state of the block.	pulse

Output Port Details

Name	Description	Type
Upward Direction	Is `true` if the input value changes towards the upward direction, else `false`.	boolean
Inflection Point	The last inflection point detected by the block.	float

Expression

`apama.analyticskit.blocks.core.Expression`

Evaluates an expression to perform arithmetic or logical calculations or string operations.

On change of input values (once all in-use inputs have been received), the expression specified in the parameter is re-calculated.

The expression language is much like EPL (see the EPL reference in the Apama documentation), but is restricted to `float`, `integer`, `string` and `boolean` types. All numeric literals are treated as `float` type values, even if they have no fractional part. Integer values can only be obtained as the result of functions such as `floor()`. Similar to EPL, `integer` and `float` are not implicitly convertible within an expression. If the result of an expression is an `integer` value, it is converted to a `float` automatically (there might be a loss of precision). Boolean values can be specified using the Boolean literals `true` and `false`. Boolean literals are case insensitive, so for example, `TRUE` and `True` are allowed. String values can be specified by enclosing string literals in double quotes, for example "my value". Special characters are encoded with a backslash (\). The following special characters (along which their encoding) are supported in string literals:

- Double quotes - `\"`
- Backslash - `\\`
- Newline - `\n`
- Tab - `\t`

The values of the inputs are available as `input1`, `input2`, `input3`, `input4` and `input5`. The input values can be of type `float`, `string`, `boolean` and `any`. Logical, relational, numerical and equality operators can be used on the values of the supported types. Logical operators are case insensitive, so for example, `AND` and `And` are allowed. Built-in methods on the `float`, `integer`, `string` and `boolean` types can be called, including `x.abs()` (absolute value of `x`), `x.pow(y)` (raise `x` to the power `y`), `x.sin()` (sine of `x` in radians), `x.round()` (rounds `x` to the nearest integer), and `s.ltrim()` (remove whitespace from the start of the string `s`). Built-in static methods of the supported types can be called by specifying the type name, followed by a dot (`.`) and the method name, for example, `float.max(input1, input2)` (find the larger of two input values). Built-in constants on the supported types can be accessed by specifying the type name, followed by a dot (`.`) and the constant name, for example, `float.E` (Euler's constant). Values of type `any` are unpacked at runtime to evaluate the expression. After unpacking, the value must be of type `float`, `string` or `boolean`. The type checker tries to validate the expressions during the validation phase, but this is not always possible with the `any` type. So if an expression contains the `any` type, even if it passes the validation phase, it can still fail at runtime due to a wrong type of variable being passed or an unsupported operation being performed. For a full list of built-in methods and constants, consult the API Reference for EPL (ApamaDoc) which can be accessed from the Apama documentation.

Some examples:

- Convert Fahrenheit to Celsius: `(input1 - 32) * 5/9`
- Convert days to seconds: `input1 * 86400`
- Average of 4 inputs: `(input1 + input2 + input3 + input4) / 4`
- Threshold comparison: `input1 > 3.1412` (but also see the Threshold block)
- Pythagoras to compute the hypotenuse of a right-angled triangle: `(input1.pow(2) + input2.pow(2)).sqrt()`
- Comparison to 3 decimal places: `(input1 * 1000 - (input1 * 1000).fractionalPart()) = (input2 * 1000 - (input2 * 1000).fractionalPart())`
- Range check: `input1 >= 1 and input1 <= 10`
- String comparison: `input1 = "my value"`
- Larger value: `float.max(input1, float.PI)`
- Remainder of integer division: `input1.round() % input2.round()`

Parameters

Name	Description	Type	Notes
Expression	An expression - a string representation of an EPL expression. Similar to EPL expressions, but with the differences as described above.	string	

Input Port Details

Name	Description	Type
input1	First input, to be used as `input1` in the expression.	any
input2	Second input, to be used as `input2` in the expression.	any
input3	Third input, to be used as `input3` in the expression.	any
input4	Fourth input, to be used as `input4` in the expression.	any
input5	Fifth input, to be used as `input5` in the expression.	any

Output Port Details

Name	Description	Type
Result	Result of the expression.	any

From Base N

`apama.analyticsbuilder.blocks.FromBaseN`

Converts a base N string to a float.

The input string can be in any integer base from 2 to 36, where letters of the English alphabet are used as digits for bases above 10. Common bases are 2 (binary), 8 (octal), 10 (decimal) and 16 (hexadecimal). The number being converted can contain a radix point.

Conversion between two arbitrary bases can be achieved by chaining this block with the To Base N block.

Parameters

Name	Description	Type	Notes
Input Base	The number base of the input stream, in the range 2 to 36.	integer	Default: 16
Radix Character	The character to use as the radix point. Expected to be a dot or a comma.	Option - one of: ■ Dot ■ Comma	Default: Dot

Input Port Details

Name	Description	Type
Base N	String input in base N.	string

Output Port Details

Name	Description	Type
Float	Numeric output.	float

KPI

`apama.analyticskit.blocks.cumulocity.KPI`

Compares a value against either a KPI (Key Performance Indicator) or the data point of a device.

This block uses data from the KPI input port or from the device which contains data points. It extracts the units, label, and the red and yellow ranges. The output indicates whether the value is within the red or yellow range specified by the KPI or data point.

The KPI input can provide properties, typically from a KPI-managed object, which include the red and yellow ranges, the unit and the label. If the device contains a data point for the specified fragment and series, then the values from the data point override those from the KPI.

Parameters

Name	Description	Type	Notes
Source Device or Device Group	The device or device group which contains a data point. If specified, then this device (typically the same device as the measurement source) is	string	Optional

Name	Description	Type	Notes
	checked to see if it contains a data point for the specified fragment and series. If it contains a data point, the red and yellow range values from the source object are used in place of the KPI values. The model editor uses the current device or asset name. This is mapped internally to the inventory identifier.		
Data Point Fragment and Series	This parameter must be specified if the Source Device or Device Group parameter is specified. It specifies a data point from the source object. This is typically the same as the fragment and series of the measurement source. The data point fragment and series must be specified as `fragment.series`.	string	Optional
Upper end of yellow range exclusive	If set, the upper end of the yellow range is treated as being exclusive.	boolean	Default: false
Upper end of red range exclusive	If set, the upper end of the red range is treated as being exclusive.	boolean	Default: false

Input Port Details

Name	Description	Type
Value	Numeric value to compare with the defined ranges.	float
KPI	Object containing the `c8y_Kpi` property.	pulse

Output Port Details

Name	Description	Type
Red	Is set to `true` when the value is in the red range (if a red range is defined).	boolean
Yellow	Is set to `true` when the value is in the yellow range (if a yellow range is defined).	boolean
Unit	The unit name from the data point.	string
Label	The label name from the data point.	string

Limit

`apama.analyticsbuilder.blocks.Limit`

Outputs a value that is kept within the defined upper and lower limits.

The input value is limited so that the output does not exceed the boundaries defined by the Lower Limit and Upper Limit parameters. If the input violates either limit, then the output is set to the parameter value, otherwise the value is passed through unchanged.

It is only mandatory to provide one of the limits. If this is the case, then the input is only limited in the direction of the specified parameter.

Parameters

Name	Description	Type	Notes
Upper Limit	Any input above this value results in the output being capped at this value.	float	Optional
Lower Limit	Any input below this value results in the output being capped at this value.	float	Optional

Input Port Details

Name	Description	Type
Value	The input value.	float

Output Port Details

Name	Description	Type
Output	The input value, if it is within the limits defined by the Lower Limit and Upper Limit parameters. If the input value exceeds one of the limit parameters, it is set to the value of that parameter.	float

Machine Learning

`apama.analyticskit.blocks.core.Zementis`

Invokes the specified Machine Learning model that scores the input data.

To use this block, the Machine Learning application needs to be available with the respective Machine Learning models in the tenant.

If the Machine Learning model does not yet exist, use the Machine Learning application to add it. If you have added the Machine Learning model while your Analytics Builder model was still in edit mode, exit the model editor and then edit your Analytics Builder model once more. This refreshes the list of available Machine Learning models and you can then select the newly added model.

Block inputs correspond to the Machine Learning model's inputs (that are marked Active) in the order specified by the Machine Learning model. All inputs used by the model must be connected. Outputs correspond to the outputs as specified by the Machine Learning model. If a PMML output is specified as "JSON", then the block outputs a string version of the JSON, but the properties of the object are also available as extra values which can be extracted using the Extract Property block, which is the recommended way of unpacking multiple values from such an output. Currently, timestamp inputs are not supported.

Parameters

Name	Description	Type	Notes
Machine Learning Model	Name of the Machine Learning model.	string	

Input Port Details

Name	Description	Type
Value 1	Input value 1.	any
Value 2	Input value 2.	any
Value 3	Input value 3.	any
Value 4	Input value 4.	any
Value 5	Input value 5.	any
Value 6	Input value 6.	any
Value 7	Input value 7.	any
Value 8	Input value 8.	any
Value 9	Input value 9.	any
Value 10	Input value 10.	any

Output Port Details

Name	Description	Type
Output 1	Output value 1.	any
Output 2	Output value 2.	any
Output 3	Output value 3.	any
Output 4	Output value 4.	any
Output 5	Output value 5.	any
Output 6	Output value 6.	any
Output 7	Output value 7.	any
Output 8	Output value 8.	any
Output 9	Output value 9.	any
Output 10	Output value 10.	any

Range

`apama.analyticsbuilder.blocks.Range`

Compares the input value against the defined lower and upper range values to detect whether the input is within or out of the range, or whether it crosses the range.

By default, the range includes the value for the lower range but excludes the value for the upper range. For example, if the lower range is 100 and the upper range is 200, then all values from 100 to 199 are within the range. 200 is considered to be out of the range.

A pulse is sent when the defined range is crossed. That is, when either the lower or upper range is crossed, or if the value goes from below the range to over the range (or vice versa) without ever being within the range.

Parameters

Name	Description	Type	Notes
Lower Range	The lower range value.	float	Optional
Include Lower Range	If selected, an input value equal to the lower range is considered to identify whether it is within or out of the range. If not selected, an input value equal to the lower range is not considered.	boolean	Default: true

Name	Description	Type	Notes
Upper Range	The upper range value.	float	Optional
Exclude Upper Range	If selected, an input value equal to the upper range is not considered to identify whether it is within or out of the range. If not selected, such an input value is considered to identify whether it is within or out of the range.	boolean	Default: true

Input Port Details

Name	Description	Type
Value	The input value to validate against the defined range.	float
Reset	Resets the state of the block.	pulse

Output Port Details

Name	Description	Type
Out of Range	Is set to `true` when the input value is not within the defined range.	boolean
In Range	Is set to `true` when the input value is within the defined range.	boolean
Crossed	Sends a pulse when the range is crossed.	pulse

Range Lookup

`apama.analyticskit.blocks.core.RangeLookup`

Finds the range in which the input value lies.

Ranges are defined using a list of unique upper bound values in increasing order. The lower bound of the first range depends on the value of the Minimum Value parameter. The lower bound of each subsequent range is defined by the upper bound of the previous range. The range to which an input value equal to a boundary value belongs depends on the Exclude Upper parameter. The block outputs the mapped value for the range the input lies within, or a failed flag if the input value is not within any of the ranges.

Parameters

Name	Description	Type	Notes
Minimum Value	The lower bound of the first range (first row). If nothing is specified, negative infinity is taken as the minimum value.	float	Optional
Ranges	A boundary and mapped value pair for the upper bound of a range for which to look up the input value and a mapped value that is to be used as the output value if the input value lies within the range.	List of Upper Bound value and Mapped Value	
Exclude Upper	If selected, an input value equal to the upper bound of a row is considered to be part of the range of the next row. If not selected, such an input value is considered to be part of the range of the current row.	boolean	Default: true
Type	The type that is to be used for the output value.	Option - one of: ■ Float ■ Boolean ■ String	Optional

Input Port Details

Name	Description	Type
Value	The input value for which the range is to be found.	float

Output Port Details

Name	Description	Type
Mapped	The Mapped Value for the range that is matched.	any
Failed	Is `true` when the input does not lie in any range. Otherwise it is `false`.	boolean

Rounding

`apama.analyticskit.blocks.core.Rounding`

Rounds the input to a specified number of decimal points or to an integer, using a selectable rule.

Rounding a numerical value means replacing it by another value that is approximately equal but has a shorter and simpler representation.

The rules available for use are:

- Up (or take the ceiling, or round towards plus infinity) rounds the input up to the nearest target number.
- Down (or take the floor, or round towards minus infinity) rounds the input down to the nearest target number.
- Towards Zero (or truncate, or round away from infinity) rounds the input towards zero to the nearest target number.
- Nearest (or round half up, or round half towards positive infinity) rounds to the nearest target number. Numbers that are equidistant from the two nearest target numbers are always rounded up. For example, value `23.5` gets rounded to `24`, but `-23.5` gets rounded to `-23.`
- Even or Nearest rounds to the nearest target number. Numbers that are equidistant from the two nearest target numbers are always rounded to the nearest even target. For example, `0.5` rounds down to `0` and `1.5` rounds up to `2.` Also known as Bankers Rounding.

The value is rounded to the nearest 'target number' - this is a whole number (if the number of decimal points is zero), or rounded to the number of decimal points specified. If the number of decimal points is negative, it is rounded to a power of 10. For example, if the number of decimal points is 2, it is rounded to the nearest 0.01 (that is, hundredths). If the number of decimal points is -3, it is rounded to the nearest 1000 (that is, thousands).

Parameters

Name	Description	Type	Notes
Rule	The rounding rule to be applied.	Option - one of: ■ Up ■ Down ■ Towards Zero ■ Nearest ■ Even or Nearest	
Number of Decimal Points	The number of decimal points the input is to be rounded to.	integer	Default: 0

Input Port Details

Name	Description	Type
Value	The input value which is to be rounded.	float

Output Port Details

Name	Description	Type
Rounded Value	The input value rounded to a specified number of decimal points or to an integer.	float

Threshold

`apama.analyticskit.blocks.core.Threshold`

Compares the input value against the defined threshold value to detect whether the input breaches the threshold or whether it crosses the threshold.

A breach occurs when the direction has been set to 'Above' and the input value is greater than the defined threshold value, or when the direction has been set to 'Below' and the input value is less than the defined threshold, or when the direction has been set to 'Above or Equal' and the input value is greater than or equal to the defined threshold value, or when the direction has been set to 'Below or Equal' and the input value is less than or equal to the defined threshold value.

A pulse is sent when the defined threshold value is crossed from any direction.

Parameters

Name	Description	Type	Notes
Threshold Value	This value is compared against the input value.	float	
Direction	The direction in which to look: whether the input value is above, below or equal to the defined threshold, or whether it crosses the defined threshold.	Option - one of: ■ Above ■ Above or Equal ■ Below ■ Below or Equal ■ Crossing	Default: Above

Input Port Details

Name	Description	Type
Value	The input value to compare against the defined threshold value.	float
Reset	Resets the state of the block.	pulse

Output Port Details

Name	Description	Type
Breached Threshold	Is set to `true` when the threshold is breached.	boolean
Within Threshold	Is set to `true` when the threshold is not breached.	boolean
Crossed Threshold	Sends a pulse when the threshold is crossed.	pulse

To Base N

`apama.analyticsbuilder.blocks.ToBaseN`

Converts a float to a base N string.

The output string can be in any integer base from 2 to 36, where letters of the English alphabet are used as digits for bases above 10. Common bases are 2 (binary), 8 (octal), 10 (decimal) and 16 (hexadecimal). The number being converted can contain a radix point. The output is calculated to a maximum precision of 16 radix places.

Conversion between two arbitrary bases can be achieved by chaining this block with the From Base N block.

Parameters

Name	Description	Type	Notes
Output Base	The number base of the output stream, in the range 2 to 36.	integer	Default: 16
Radix Character	The character to use as the radix point. Expected to be a dot or a comma.	Option - one of: ■ Dot ■ Comma	Default: Dot

Input Port Details

Name	Description	Type
Float	Numeric input.	float

Output Port Details

Name	Description	Type
Base N	String output in base N.	string

Aggregate

Average (Mean)

`apama.analyticskit.blocks.core.Mean`

Calculates the mean of the values over time.

This block is suitable for continuous values, even if they are irregularly sampled. The time between inputs or samples is significant, while the number of samples is not. Use this block, for example, if the input is a physical property, such as temperature, that is sampled either regularly or irregularly (for example, only generating measurement values on a change in temperature). Use the Discrete Statistics block instead of the Average (Mean) block for independent measurements, such as ticket sales, where the number of measurements is significant, but the time between measurements is not.

The mean is defined as the sum of the input's value multiplied by how long the input has stayed at that value, within an optional window, divided by the window duration or the time since the block was started or last reset, whichever is smallest.

The block can operate over a time-bounded window that is specified with the Window Duration parameter. If this parameter is not specified, the block uses an unbounded window. The Reset input port clears the window contents. Output is generated on any new input or, if the Output Threshold parameter is set, only when the output changes by more than the specified output threshold (which includes if no further input occurs, or the value only changes due to old entries expiring). The Sample input port can be used to force re-evaluation and generate the latest value.

See also the "Value types" topic in the Analytics Builder documentation for more details and an example of the frequency of output from this block and how values in windows behave.

If a window is configured, the block uses a set of 20 buckets, so the expired value is an approximation of the average value across a bucket.

Note:The Average (Mean) block generates the mean for an individual device. If the input comes from a device group, the mean is generated separately for each device in that group. To calculate

and generate aggregate values for the group as a whole (not for individual devices), use the Group Statistics block.

Parameters

Name	Description	Type	Notes
Window Duration (secs)	If present, the amount of time (in seconds) for which values are to be kept in the window. This must be a finite and positive number.	float	Optional
Output Threshold	If present, the output to be sent at the point when it changes by at least this value. This must be a finite and positive number.	float	Optional

Input Port Details

Name	Description	Type
Value	Input for which the mean is to be calculated.	float
Reset	Clears the content of the window.	pulse
Sample	Forces re-evaluation of the current mean value and sends the output.	pulse

Output Port Details

Name	Description	Type
Average	The sum of the value multiplied by how long it stays at that value divided by the total time.	float

Counter

`apama.analyticskit.blocks.core.Counter`

Gives a count of the total inputs and repeated inputs.

If two consecutive input values have different types, they will not be evaluated as repeat values. All other evaluations for if two values are equal to each other follow the same rules as EPL. For more information, see the following section in the Apama documentation: Developing Apama Applications > EPL Reference > Types.

Input Port Details

Name	Description	Type
Value	The input to be counted.	any
Reset	Reset the counters.	boolean

Output Port Details

Name	Description	Type
Count	The total count of all the inputs.	float
Number Same	The current count of repeated inputs with the same value. On a change of value, this is reset to one.	float

Discrete Statistics

`apama.analyticsbuilder.blocks.DiscreteStatistics`

Generates statistics of sum, count, average (mean), standard deviation, minimum and maximum for discrete input values.

This block is suitable for discrete time inputs, where the number of samples (or inputs) is significant, while the time between them is not. The Average (Mean) and Standard Deviation blocks are more suitable for continuous values that may be irregularly sampled, such as temperature readings. Use this block, for example, if each sample represents a transaction such as a ticket being sold.

If the Sample input port is not connected, every value is sampled, including during the same activation period as a reset. As every value is used, the standard deviation uses the generic formula: $\sigma^2 = \sum(x - \mu)^2 / N$.

If the Sample input port is connected, the block only samples the data when the Sample input port receives a signal. In this case, the sampling standard deviation uses the formula: $\sigma^2 = \sum(x - \mu)^2 / (N-1)$.

If reset and sample signals are received together, the reset is processed first.

Input Port Details

Name	Description	Type
Value	The current value.	float
Sample	Use the current value in generating statistics. If left unconnected, all values are used.	pulse

Name	Description	Type
Reset	Reset the state of the block.	pulse

Output Port Details

Name	Description	Type
Sum	Sum of the sampled input values, $\sum x$.	float
Count	Count of the number of sampled input values, N.	float
Average	Average (mean) of the sampled input values, $\mu = \sum x / N$.	float
Standard Deviation	Standard deviation of the sampled input values. If all values are being sampled, with the Sample input port disconnected, the generic standard deviation, $\sigma^2 = \sum(x - \mu)^2 / N$, is used. Otherwise the sampling standard deviation, $\sigma^2 = \sum(x - \mu)^2 / (N-1)$, is used.	float
Minimum	Minimum of the sampled input values.	float
Maximum	Maximum of the sampled input values.	float

Gradient

`apama.analyticskit.blocks.core.Gradient`

Calculates the weighted linear regression gradient for the values.

A gradient measures the rate of change of a value over time. A positive gradient indicates an increase of the input values, and a negative gradient indicates a decrease of the input values. The magnitude of the gradient signifies the scale of change.

The block can operate over a time-bounded window that is specified with the Window Duration parameter. If this parameter is not specified, it uses an unbounded window and the block re-evaluates for every 1 second, and will use 1-second buckets. If a window is configured, the block will use a set of 20 buckets, so the time of expired values is an approximation to the nearest bucket interval. The first gradient output is generated only when a minimum of two buckets is available for computation.

The Reset input clears the content of the window. Sample input can be used to force re-evaluation and generate the latest value.

Parameters

Name	Description	Type	Notes
Window Duration (secs)	If present, the amount of time (in seconds) for which values are to be kept in the window. This must be a finite and positive number.	float	Optional

Input Port Details

Name	Description	Type
Value	The input value for which the gradient is to be calculated.	float
Reset	Clears the content of the window.	pulse
Sample	Forces re-evaluation of the current value and sends the output.	pulse

Output Port Details

Name	Description	Type
Gradient	The gradient of the input values.	float

Group Statistics

`apama.analyticskit.blocks.core.GroupStatistics`

Generates periodic aggregate values across all the devices in a group for which the block has received input values.

This block generates the following aggregate values:

- Minimum
- Maximum
- Device Count
- Average
- Standard Deviation
- Variance

The block can operate over a time-bounded window that is specified with the Window Duration parameter. If this parameter is not specified, the block uses an unbounded window. Output is generated periodically as specified by the Output Period parameter.

If a window is configured, the block uses a set of 20 buckets, so the expired value is an approximation to the nearest bucket interval.

Note: The Group Statistics block calculates and generates aggregate values for the group as a whole (not for individual devices). To generate aggregates for an individual device in a group, use the Average (Mean), Standard Deviation, or Minimum/Maximum blocks. The Group Statistics block only considers devices from which it has received input values.

Parameters

Name	Description	Type	Notes
Window Duration (secs)	If present, the amount of time (in seconds) for which values are to be kept in the window. This must be a finite and positive number.	float	Optional
Output Period (secs)	The amount of time (in seconds) between each output. This must be a finite and positive number.	float	Default: 5.0

Input Port Details

Name	Description	Type
Value	Input value for which the aggregate values are to be calculated.	float

Output Port Details

Name	Description	Type
Minimum	The smallest input value (closest to negative infinity) across all the devices in a group.	float
Maximum	The largest input value (closest to positive infinity) across all the devices in a group.	float
Device Count	The number of devices in a group for which input values have been received so far.	float
Average	The sum of the value multiplied by how long it stays at that value divided by the total time and the device count.	float

Name	Description	Type
Standard Deviation	The standard deviation of the input values across all the devices in a group.	float
Variance	The variance of the input values across all the devices in a group.	float

Integral

`apama.analyticskit.blocks.core.Integral`

Calculates the integral of the input value over time.

Integral is defined as the sum of the input's value multiplied by how long the input has stayed at that value, within an optional window, since the block was started or last reset.

The block can operate over a time-bounded window that is specified with the Window Duration parameter. If this parameter is not specified, it uses an unbounded window. The Reset input clears the window contents. Output is generated on any new input or, if the Output Threshold parameter is set, only when the output changes by more than the specified output threshold (which includes if no further input occurs, or the value only changes due to old entries expiring). The Sample input can be used to force re-evaluation and generate the latest value.

If a window is configured, the block will use a set of 20 buckets, so the expired value is an approximation of the average value across a bucket.

Parameters

Name	Description	Type	Notes
Window Duration (secs)	If present, the amount of time (in seconds) for which values are to be kept in the window. This must be a finite and positive number.	float	Optional
Output Threshold	If present, the output to be sent at the point when it changes by at least this value. This must be a finite and positive number.	float	Optional

Input Port Details

Name	Description	Type
Value	Input for which the integral is to be calculated.	float
Reset	Clears the content of the window.	pulse

Name	Description	Type
Sample	Forces re-evaluation of the current integral value and sends the output.	pulse

Output Port Details

Name	Description	Type
Integral Value	The sum of the value multiplied by how long it stays at that value.	float

Minimum / Maximum

`apama.analyticskit.blocks.core.MinMax`

Calculates the minimum and maximum of a value over time.

The minimum is defined as the smallest value (closest to negative infinity) of the input values in the window, and the maximum is defined as the largest value (closest to positive infinity) of the input values in the window.

The block can operate over a time-bounded window that is specified with the Window Duration parameter. If this parameter is not specified, the block uses an unbounded window. The Reset input clears the window contents. Output is generated on any new input that exceeds the current minimum or maximum, or, if the Window Duration parameter is set, when a previous minimum or maximum expires.

If a window is configured, the block uses a set of 20 buckets, so the time of expired values is an approximation to the nearest bucket interval.

Note: The Minimum/Maximum block generates the minimum and maximum for an individual device. If the input comes from a device group, these values are generated separately for each device in that group. To calculate and generate aggregate values for the group as a whole (not for individual devices), use the Group Statistics block.

Parameters

Name	Description	Type	Notes
Window Duration (secs)	If present, the amount of time (in seconds) for which values are to be kept in the window. This must be a finite and positive number.	float	Optional

Input Port Details

Name	Description	Type
Value	Input for which the minimum and maximum is to be calculated.	float
Reset	Clears the content of the window.	pulse

Output Port Details

Name	Description	Type
Minimum	The smallest value in the window (closest to negative infinity).	float
Maximum	The largest value in the window (closest to positive infinity).	float

Standard Deviation

`apama.analyticskit.blocks.core.StandardDeviation`

Calculates the standard deviation and variance of the values over time.

This block is suitable for continuous values, even if they are irregularly sampled. The time between inputs or samples is significant, while the number of samples is not. Use this block, for example, if the input is a physical property, such as temperature, that is sampled either regularly or irregularly (for example, only generating measurement values on a change in temperature). Use the Discrete Statistics block instead of the Standard Deviation block for independent measurements, such as ticket sales, where the number of measurements is significant, but the time between measurements is not.

Standard deviation is a measure that is used to quantify the amount of variation or dispersion of a set of data values. A low standard deviation indicates that the data points tend to be close to the mean of the set, while a high standard deviation indicates that the data points are spread out over a wider range of values.

The block can operate over a time-bounded window that is specified with the Window Duration parameter. If this parameter is not specified, the block uses an unbounded window. The Reset input port clears the window contents. The Sample input port can be used to force re-evaluation and generate the latest value. Output is generated on any new input or, if the Window Duration parameter is set, output is generated periodically on every new bucket that is added to the window.

If a window is configured, the block uses a set of 20 buckets, so the time of expired values is an approximation to the nearest bucket interval.

Note: The Standard Deviation block generates the standard deviation and variance for an individual device. If the input comes from a device group, these values are generated separately for each

device in that group. To calculate and generate aggregate values for the group as a whole (not for individual devices), use the Group Statistics block.

Parameters

Name	Description	Type	Notes
Window Duration (secs)	If present, the amount of time (in seconds) for which values are to be kept in the window. This must be a finite and positive number.	float	Optional

Input Port Details

Name	Description	Type
Value	Input for which the standard deviation and variance is to be calculated.	float
Reset	Clears the content of the window.	pulse
Sample	Forces re-evaluation of the current value and sends the output.	pulse

Output Port Details

Name	Description	Type
Standard Deviation	The standard deviation of the input values.	float
Variance	The variance of the input values.	float

Flow Manipulation

Combiner

`apama.analyticskit.blocks.core.Combiner`

Calculates the output based on the selected mode and the connected inputs.

Available modes are:

- Minimum: Outputs the minimum of the connected inputs which have received a value. All numeric literals are treated as `float` type.

- Maximum: Outputs the maximum of the connected inputs which have received a value. All numeric literals are treated as `float` type.
- Average (Mean): Outputs the average (mean) of the connected inputs for which a value has been received. All numeric literals are treated as `float` type.
- Latest: Outputs the latest changed value. If multiple values change in a single activation, then the input port with the highest number is used. For example, if Value 1 and Value 2 get an updated value, Value 2 is selected for output. Inputs must be of the same type.

Parameters

Name	Description	Type	Notes
Mode	The mode to be selected.	Option - one of: - Minimum - Maximum - Average (Mean) - Latest	

Input Port Details

Name	Description	Type
Value 1	First input value to the block.	any
Value 2	Second input value to the block.	any
Value 3	Third input value to the block.	any
Value 4	Fourth input value to the block.	any
Value 5	Fifth input value to the block.	any

Output Port Details

Name	Description	Type
Combined Value	The calculated combined value.	sameAsAll(value1, value2, value3, value4, value5)

Gate

`apama.analyticskit.blocks.core.Gate`

Blocks the input from going to output unless the gate is open and enabled.

The block will start disabled if the Enable input is connected, otherwise the block will always be enabled. The block will start closed if the Open input is connected, otherwise the block will start open.

Parameters

Name	Description	Type	Notes
Null Value	The value to use as output value when the gate is either disabled or closed. The exact type of the value depends on the Null Value Type parameter. If the Null Value Type parameter is specified then the string value is parsed as the specified type. If the Null Value Type parameter is not specified, and the value is parsable into a float or a boolean value then it is parsed into the corresponding value type, otherwise it is used as a string value.	string	Optional
Null Value Type	The type of the value specified by the Null Value parameter. If specified, the null value is parsed into the specified type.	Option - one of: ■ Float ■ Boolean ■ String	Optional
Close Duration (secs)	The amount of time (in seconds) the gate should be closed when a close signal is received. If the parameter is not specified, then after a close signal is received, the gate remains closed until an open signal is received. If the parameter is specified, then the gate automatically opens after the specified number of seconds, unless another open or close signal is received before the time has elapsed. On opening, if the gate is also enabled, then the latest input value is sent out as output. This must be a finite and positive number.	float	Optional

Input Port Details

Name	Description	Type
Value	The value to pass to output.	any
Enable	Enables the gate. If the gate is always to be enabled, then leave this port unconnected.	boolean
Open	Opens the gate. If the gate is to be open at the start, then leave this port unconnected.	pulse
Close	Closes the gate. If the Close Duration parameter is specified, then the gate is closed only for the specified number of seconds.	pulse

Output Port Details

Name	Description	Type
Gated Value	Same as the input value when the gate is open and enabled. Otherwise the value specified by the Null Value parameter.	input(value)
Activated	Output pulse. Generated when the gate becomes active. The gate is active when it is both open and enabled.	pulse
Deactivated	Output pulse. Generated when the gate becomes inactive. The gate is inactive when it is either closed or disabled.	pulse

Latch Values

`apama.analyticskit.blocks.core.Latch`

Latches the latest input value received while the block is enabled.

Only generates an output if the input value changes and the block is enabled. The block will start disabled if the Enable input is connected, otherwise the block will always be enabled.

Input Port Details

Name	Description	Type
Value	The input value to be monitored.	any
Enable	Enables the block. If the block is always to be enabled, then leave this unconnected.	boolean

Output Port Details

Name	Description	Type
Latched Value	Same as the input value. Generated only if the input has changed while the block was enabled.	input(value)
Changed	Output pulse. Generated only if the input has changed while the block was enabled.	pulse
Enabled	Output pulse. Generated when the block is enabled.	pulse
Disabled	Output pulse. Generated when the block is disabled.	pulse

Pulse

`apama.analyticskit.blocks.core.Pulse`

Converts a non-pulse input into a pulse output.

This is useful with blocks which consume both pulse and non-pulse values, and where the input value is treated as non-pulse without the explicit conversion.

For example, a numeric value passed to the OR block is treated as `true` if non-zero (as described in the "Type conversions" topic of the Analytics Builder documentation). However, when passing a numeric value to the Pulse block and then connecting the output of the Pulse block to the OR block, the numeric value is converted to a pulse so the OR block sends a pulse. The block can be configured to send a pulse if the value changes (default pulse conversion behavior), on every input, or on every non-zero value.

Parameters

Name	Description	Type	Notes
Mode	Controls when the block sends a pulse.	Option - one of: ■ On value change ■ On every input ■ On non-zero values	Default: On value change

Input Port Details

Name	Description	Type
Value	The input to treat as pulse.	any

Output Port Details

Name	Description	Type
Pulse	The output pulse value converted from the input value.	pulse

Selector

`apama.analyticsbuilder.blocks.Selector`

Outputs a parameter value depending on which input port has a `true` value, lowest number taking precedence.

You specify the output value that is to be sent using the parameters of this block. Only one of the parameter values is sent in the output. It is sent when the corresponding input port receives a `true` value. If more than one input port receives a `true` value, then the input port is used which has the lowest number in its name. For example, Input 1 has a higher priority than Input 2.

If all input values are `false`, then the value specified with the No Input parameter is sent.

Example: Input 1 has "high", Input 2 has "medium", Input 3 has "low", and the No Input parameter has the value "off". If none of the input ports receives a `true` value, then "off" is sent as the output value. If both the Input 2 and Input 3 ports receive a `true` value and Input 1 receives a `false` value, then "medium" is sent as the output value. This is because Input 2 has a higher priority than Input 3.

Parameters

Name	Description	Type	Notes
Input 1	The output to be sent when the Input 1 port receives a `true` value.	string	
Input 2	The output to be sent when the Input 2 port receives a `true` value, but only if this is the port with the lowest number in its name.	string	Optional
Input 3	The output to be sent when the Input 3 port receives a `true` value, but only if this is the port with the lowest number in its name.	string	Optional

Name	Description	Type	Notes
Input 4	The output to be sent when the Input 4 port receives a `true` value, but only if this is the port with the lowest number in its name.	string	Optional
Input 5	The output to be sent when the Input 5 port receives a `true` value, but only if this is the port with the lowest number in its name.	string	Optional
No Input	The output to be sent when there is no input which has a `true` value.	string	
Type	The type that is to be used for the output value.	Option - one of: ■ Float ■ Boolean ■ String	Optional

Input Port Details

Name	Description	Type
Input 1	Causes the Input 1 parameter value to be sent if `true`.	boolean
Input 2	Causes the Input 2 parameter value to be sent if `true` (and no lower numbered input is `true`).	boolean
Input 3	Causes the Input 3 parameter value to be sent if `true` (and no lower numbered input is `true`).	boolean
Input 4	Causes the Input 4 parameter value to be sent if `true` (and no lower numbered input is `true`).	boolean
Input 5	Causes the Input 5 parameter value to be sent if `true` (and no lower numbered input is `true`).	boolean

Output Port Details

Name	Description	Type
Output	The output value from one of the parameters.	any

Switch

`apama.analyticsbuilder.blocks.Switch`

Outputs the values from a given input, or acts as a circuit breaker.

If the Selected Input parameter is specified, then the given input must exist and the corresponding input port must be connected.

If the Selected Input parameter is not specified, then the block acts as a circuit breaker.

You can use the initial default names for the inputs. However, you can also rename them to be more descriptive. The input names must be unique, so two inputs cannot share the same name. Connected inputs must all be of the same type.

The expected use case is that the Selected Input parameter is set to a template parameter which can then be set individually for each model instance to decide which input is used.

Parameters

Name	Description	Type	Notes
Selected Input	To specify an output, specify one of the five input names. To act as a circuit breaker, leave this parameter empty.	string	Optional
Name for input1	The name of the first input to the block.	string	Default: input1
Name for input2	The name of the second input to the block.	string	Default: input2
Name for input3	The name of the third input to the block.	string	Default: input3
Name for input4	The name of the fourth input to the block.	string	Default: input4
Name for input5	The name of the fifth input to the block.	string	Default: input5

Input Port Details

Name	Description	Type
input1	The first input.	any
input2	The second input.	any
input3	The third input.	any
input4	The fourth input.	any
input5	The fifth input.	any

Output Port Details

Name	Description	Type
Switch	The value from the specified input, or no value (circuit breaker) if the Selected Input parameter is empty.	sameAsAll(input1, input2, input3, input4, input5)

Time Delay

`apama.analyticskit.blocks.core.TimeDelay`

Delays the input by the specified amount of time.

Parameters

Name	Description	Type	Notes
Delay (secs)	The amount of time (in seconds) by which the input will be delayed. This must be a finite and positive number.	float	

Input Port Details

Name	Description	Type
Value	The input value to be delayed.	any

Output Port Details

Name	Description	Type
Delayed Value	The delayed output value.	input(value)

Utility

Constant Value

`apama.analyticsbuilder.blocks.ConstantValue`

Outputs a value, either when the Trigger input port receives a signal or at startup.

The Trigger input port can be used to delay the output until a trigger input is received. If the Trigger input port is not connected, then the block outputs a value when the model is activated, which may trigger further processing.

The Value parameter can be treated as either a string, or as a JSON value. For JSON string, number or boolean, then that will be the type of the output. For JSON objects, the output will be a pulse with the properties from the object. JSON arrays are only permitted within an object.

Parameters

Name	Description	Type	Notes
Value	The value to output.	string	
Type	How to interpret the Value parameter.	Option - one of: ■ string ■ JSON	Default: JSON

Input Port Details

Name	Description	Type
Trigger	If connected, send output on this trigger.	boolean

Output Port Details

Name	Description	Type
Output	The output value from the Value parameter.	any

Cron Timer

`apama.analyticskit.blocks.core.CronTimer`

Sends a signal output based on cron-like periodic timer syntax.

The timer tick output is sent every time the time matches the pattern. Each parameter can be set to an asterisk (*) to match all times, a list of comma-separated numbers to fire at those specific times, or "* / number" to fire on every multiple of the number.

For example:

To send a signal every Wednesday at 20:30:

- Day of Week: 4
- Hour: 20

- Minute: 30

To send a signal every 5 seconds:

- Seconds: * / 5

The Days of Month and Days of Week parameters operate together such that if both are specified, then any match in either parameter triggers an output. For example, if Days of Month is set to 15 and Days of Week is set to 1, then there is an output on every Monday of the month and the 15th regardless of which day that is.

This block is not supported in simulation mode.

Parameters

Name	Description	Type	Notes
Seconds	The seconds at which to trigger the signal. Range: 0 to 59.	string	Default: 0
Minutes	The minutes at which to trigger the signal. Range: 0 to 59.	string	Default: *
Hours	The hours at which to trigger the signal. Range: 0 to 23.	string	Default: *
Days of Month	The days of the month on which to trigger the signal. Range: 1 to 31.	string	Default: *
Months	The months in which to trigger the signal. Range: 1 to 12.	string	Default: *
Days of Week	The days of the week on which to trigger the signal. Range: 0 (Sunday) to 6 (Saturday).	string	Default: *

Output Port Details

Name	Description	Type
Timer Tick	Sent at each trigger time based on the cron parameters.	pulse

Duration

`apama.analyticskit.blocks.core.Duration`

Measures the time elapsed from a set start time.

The start time is set by a start signal which activates the block. If the block is already active, then a start signal is ignored and the existing measurement remains unaffected. The block is deactivated with a reset signal which also disables any periodic outputs.

The block generates a float output of the time elapsed since the start signal, measured in seconds. If the block is inactive at the time it receives a measure signal, then 0.0 is generated as the output.

If multiple signals are received at the same time, they are processed in the order of measure, reset and start. Thus, for example, if measure and reset signals are received together, the block first generates an output with the current duration and then resets its state. Processing the inputs in this order allows the current state of the block to be output and the block to be restarted within a single unit of time, if desired.

Parameters

Name	Description	Type	Notes
Periodic Output (secs)	The amount of time (in seconds) between automatically generated outputs by an active block. If not set, then an output is only generated when a measure signal is received. This must be a finite and positive number.	float	Optional

Input Port Details

Name	Description	Type
Measure	Triggers an output of the duration (in seconds) since the block was activated. If the block is inactive, then 0.0 is output.	pulse
Reset	Deactivates and resets the state of the block.	pulse
Start	Activates the block.	pulse

Output Port Details

Name	Description	Type
Duration	The time in seconds since a start signal was received.	float

Extract Property

`apama.analyticskit.blocks.core.ExtractProperty`

Extracts the specified property from the input value and converts it to the specified type.

The value in the Value input named by the Property Path parameter should be a string, number or boolean.

You can specify a period (.) as part of the Property Path parameter to extract nested values from a dictionary.

For example: If the input is `{ "location" : { "city" : "Cambridge" } }` (in JSON form), then you can extract that value by specifying `location.city` as the Property Path parameter.

You can also specify square brackets as part of the Property Path parameter to extract a specific element from a sequence.

If the value is an object, then the properties of that object are output as properties on the Extracted Value output port.

For example: If the input is `{ "users" : [ { "age" : 40.375 } ] }` (in JSON form), then you can extract that value by specifying `users[0].age` as the Property Path parameter.

The block does not support extracting entries from a dictionary whose key contains special characters like the period (.) or square brackets. Also it does not support extracting entries from a sequence without using square brackets, for example, `users.0.id` must be written as `users[0].id`.

In converting a string to a float, this block treats an empty string as a value of 0.0, rather than as not parseable.

Parameters

Name	Description	Type	Notes
Property Path	The name or path of the property that is to be extracted from the input value. If not set, all of the properties are output.	string	Optional
Property Type	The type to which the property value is to be converted. If set to Properties, a pulse is output with properties from the extracted value.	Option - one of: ■ String ■ Boolean ■ Float ■ Properties	Default: String
Clear On Missing	If selected, the default value of the specified type is output if the Property Path parameter was not	boolean	Default: false

Name	Description	Type	Notes
	specified or if the value cannot be converted into the specified type.		

Input Port Details

Name	Description	Type
Value	The input value from which the property is to be extracted.	any

Output Port Details

Name	Description	Type
Extracted Value	The value that has been extracted from the input value.	any

Geofence

`apama.analyticsbuilder.blocks.Geofence`

Compares the input value against the defined geofence value to detect whether the device is within the geofence, and whether the device entered or exited the geofence.

The boundary line itself is considered to be outside of the geofence area.

Parameters

Name	Description	Type	Notes
Geofence	A polygon describing an area on the Earth's surface. Note that crossing the international date line, namely the longitude of +/- 180 degrees, is not supported.	sequence	

Input Port Details

Name	Description	Type
Position	The input value from which the properties for the longitude (lng) and latitude (lat) are to be extracted.	any

Output Port Details

Name	Description	Type
Within Geofence	Is set to `true` when the device is within the defined geofence.	boolean
Entered Geofence	Sends a signal when the device enters the defined geofence.	pulse
Exited Geofence	Sends a signal when the device leaves the defined geofence.	pulse

Missing Data

`apama.analyticskit.blocks.core.MissingData`

Generates an output if the input has not occurred for a set amount of time.

The block can optionally detect an absence of changes in value if repeated same-value inputs are not counted.

Parameters

Name	Description	Type	Notes
Ignore Repeated Inputs	If selected, same-value inputs are ignored.	boolean	Default: false
Duration (secs)	The amount of time (in seconds) the block waits for an input. This must be a finite and positive number.	float	

Input Port Details

Name	Description	Type
Value	The data whose presence or absence is being detected.	any
Reset	Resets the state of the block - resets any timer and the previous value.	pulse

Output Port Details

Name	Description	Type
Missing Data	Absence of data, `true` if no input is received within the specified duration, else `false`.	boolean

Set Properties

`apama.analyticsbuilder.blocks.SetProperties`

Outputs a pulse with properties set from values on the input ports.

The property names are taken from the parameters and the values from the input ports. New properties are only output if they have been received. An output is sent as soon as at least one of the inputs is provided.

Properties are set on the output in the following order of precedence:

1. Any properties which have been explicitly specified by the use of a parameter. If an input is a pulse, it is treated as an object using the properties of that input. If an input has a primary value (not a pulse), then the primary value is used. To use the properties instead, use the Extract Property block with the Property Path parameter not set and the Property Type parameter set to Properties which replaces the primary value with a pulse (which is ignored) and the properties are used.
2. The properties of any `Value` object on an input port which does not have the corresponding parameter set. This is a straight merge of the `properties` dictionary. If two dictionaries have the same property key, then the input port with the lowest identifier has precedence. Thus any shared properties on Input 1 overwrite properties from Input 2 and down.
3. Any properties on a `Value` object provided to the Merge input port are kept if they are not overwritten by either of the operations above. The optional Merge input port allows chaining or supplementing a set of properties from another block.

Thus, any properties set on an input are overwritten by those with the same name on a higher precedence input, or when an input is configured for the specified property.

Parameters

Name	Description	Type	Notes
Property 1	A property to set in the output, using the value from input port 1.	string	Optional
Property 2	A property to set in the output, using the value from input port 2.	string	Optional

Name	Description	Type	Notes
Property 3	A property to set in the output, using the value from input port 3.	string	Optional
Property 4	A property to set in the output, using the value from input port 4.	string	Optional
Property 5	A property to set in the output, using the value from input port 5.	string	Optional

Input Port Details

Name	Description	Type
Input 1	Value to be added using the property name in parameter Property 1.	any
Input 2	Value to be added using the property name in parameter Property 2.	any
Input 3	Value to be added using the property name in parameter Property 3.	any
Input 4	Value to be added using the property name in parameter Property 4.	any
Input 5	Value to be added using the property name in parameter Property 5.	any
Merge	Source to merge with the specified properties. Properties from here not replaced by an input value are passed through to the output.	any

Output Port Details

Name	Description	Type
Output	The output value with extra properties supplied.	pulse

Text Substitution

`apama.analyticsbuilder.blocks.TextSubstitution`

Substitutes identifiers marked with a hash and braces (for example, `#{name}`) in the text template with corresponding entries from the input values.

At least one of the Object or Source input ports must be connected. Identifiers that cannot be resolved are not substituted.

The identifiers prefixed with `source.` (for example, `#{source.name}`) are searched for in the value received on the Source input port. For example, if the value received on the Source input port is `{ "name": "sample_name" }`, then the identifier `#{source.name}` is resolved to `sample_name`.

The identifiers not prefixed with `source.` are searched for in the value received on the Object input port.

Nested identifiers can be specified by separating them with a dot (.). For example, when the Object input port has received the value `{ "address": { "street": { "name": "example_street" }}}`, then the identifier `#{address.street.name}` is resolved to `example_street`.

Keys with a dot (.) in them are not supported, so if the Object input port value is of the form `{ "address.street": { "name": "example_street" }}`, then the `example_street` value cannot be resolved because the identifier `#{address.street.name}` expects `street` to be nested inside the `address` entry.

Primitive values such as `integer`, `float`, `boolean` and `string` are substituted directly, but complex values are converted to a JSON representation before substitution.

Any identifier with the text `time` (case-insensitive) in it and value type `float` is interpreted as a timestamp value and is converted into the format `yyyy-MM-ddTHH:mm:ss.SSSZ` before substitution. This can be modified with optional parameters using the following syntax: `{time:param1="value1",param2="value2"}`. Use the parameter `TZ="time_zone"` to specify a different time zone and/or use the parameter `FORMAT="format_string"` to specify a different format. For example, `#{time:TZ="America/New_York",FORMAT="HH:mm:ssZ"}` specifies the time zone for New York and the format to be `HH:mm:ssZ`. The model fails to activate if the time zone is not recognized or the format is invalid.

Note: The format string for the time must not contain quotes (") and braces ({ and }).

A hash (#) can be specified in the text template by escaping it as follows: `#{#}`.

For more information, see the list of time zones and the list of valid time format strings in the Apama documentation.

Parameters

Name	Description	Type	Notes
Text Template	The text that is used to generate the output by substituting the identifiers in it, such as #{name}, with the values from the input ports.	string	

Input Port Details

Name	Description	Type
Object	Used to substitute identifiers that are not prefixed with `source.`. For example, `#{name}` or `#{timestamp}`.	any

Name	Description	Type
Source	Used to substitute identifiers that are prefixed with `source.`. For example, `#{source.name}`.	any

Output Port Details

Name	Description	Type
Output	String containing the substitutions from the text template.	string

Toggle

`apama.analyticskit.blocks.core.Toggle`

Converts two pulse inputs to a boolean output based on the set and reset signals, with optional delays.

Without delays, the output state is changed to `true` on a set signal, and changed to `false` on a reset signal. If both signals are received at the same time, the output state is toggled (to `true` if signals are received for the first time).

If delay times are specified, the output state is only changed to `true` or `false` after the delay has been applied.

The following exception applies if both signals are received at the same time: the output state is only toggled if both delay times are the same, or have not been specified at all.

Parameters

Name	Description	Type	Notes
Set Delay (secs)	The amount of time (in seconds) after which the set signal is processed. If the parameter is not specified, then the signal is immediately processed. This must be a finite and positive number.	float	Optional
Reset Delay (secs)	The amount of time (in seconds) after which the reset signal is processed. If the parameter is not specified, then the signal is immediately processed. This must be a finite and positive number.	float	Optional

Input Port Details

Name	Description	Type
Set	Sets the output to `true`. Any pending delayed set or reset signals are cancelled on the new input.	pulse
Reset	Sets the output to `false`. Any pending delayed set or reset signals are cancelled on the new input.	pulse

Output Port Details

Name	Description	Type
Value	The output value generated from the input signals.	boolean