SOFTHUB CONFIGURATION

Configuring the OmnIoT SoftHub Rule Engine

Rel200515

www.omniot.com

Table of Contents

1. Introduction 1

1.1 SoftHub Platform Components 1

1.2 SoftHub Rule Engine Mechanics 2

1.3 Configuration File Contents and Deployment 4

2. Using the SoftHub Configuration Utility 5

2.1 The Toolbar Menu 6

2.2 Setting the Target Platform 6

2.3 The Configuration Options Panel 7

2.4 The Help Panel 9

2.5 The Current Object Panel 9

2.6 The Object Info Panel 10

3. SoftHub System Options 11

3.1 Hub Options 11

3.2 Ble Settings 13

3.3 Email Settings 15

3.4 GPIO Settings 16

4. SoftHub Rule Engine Objects 17

4.1 Sensor Stream Objects 17

4.1.01 AntSensorStream Sensor Stream object 18

4.1.02 BleSensorStream Sensor Stream object 19

4.2 User Report Objects 21

4.2.01 UserReport User Report object (Main) 21

4.2.02 UserReport User Report object (Sensor Stream Data) 22

4.2.03 UserReport User Report object (Sensor Data Average) 23

4.2.04 UserReport User Report object (Sensor Data Cache) 24

4.2.05 UserReport User Report object (GPIO Pin Values) 25

4.3 Event Objects 26

4.3.01 AppStarting Event object 26

4.3.02 AppStopping Event object 27

4.3.03 UserDefinedEvent Event object 27

4.3.04 SensorConnect Event object 28
4.3.05 SensorDisconnect Event object 28
4.3.06 NewSensorPacket Event object 29
4.3.07 GpioPinLow Event object 29
4.3.08 GpioPinHigh Event object 30
4.3.09 TimerExpired Event object 30
4.4 Action Objects 31
4.4.01 EnableSensorStream Action object 31
4.4.02 DisableSensorStream Action object 32
4.4.03 EnableBleNotifications Action object 32
4.4.04 DisableBleNotifications Action object 33
4.4.05 InitiateBleRead Action object 34
4.4.06 InitiateBleWrite Action object 34
4.4.07 StartOneshotTimer Action object 35
4.4.08 StartContinuousTimer Action object 36
4.4.09 StartOneShotTodTimer Action object 37
4.4.10 StartContinuousTodTimer Action object 38
4.4.11 StopTimer Action object 39
4.4.12 SendData Action object 39
4.4.13 SendReport Action object 40
4.4.14 PublishData Action object 41
4.4.15 PublishReport Action object 42
4.4.16 LogData Action object 43
4.4.17 LogReport Action object 43
4.4.18 SendEmail Action object 44
4.4.19 Initialize Flag Action object 45
4.4.20 SetFlag Action object 45
4.4.21 ClearFlag Action object 46
4.4.22 ToggleFlag Action object 47
4.4.23 InitializeCounter Action object 47
4.4.24 IncrementCounter Action object 48
4.4.25 DecrementCounter Action object 49
4.4.26 InitDataAverager Action object 50

4.4.27 UpdateDataAverager Action object 51
4.4.28 InitSensorDataCache Action object 51
4.4.29 UpdateSensorDataCache Action object 52
4.4.30 SetGpioPinLow Action object 53
4.4.31 SetGpioPinHigh Action object 53
4.4.32 ToggleGpioPin Action object 54
4.4.33 ExecSysCommand Action object 55
4.4.34 EnableHubAction Action object 56
4.4.35 EnableHubEvent Action object 56
4.4.36 EnableHubRule Action object 57
4.4.37 DisableHubAction Action object 58
4.4.38 DisableHubEvent Action object 58
4.4.39 DisableHubRule Action object 59
4.4.40 ToggleHubAction Action object 60
4.4.41 ToggleHubEvent Action object 61
4.4.42 ToggleHubRule Action object 61
4.4.43 SoftHubStop Action object 62
4.5 State Objects 63
4.5.01 SensorConnected State object 63
4.5.02 SensorDataValue State object 64
4.5.03 SensorDataAveragerValue State object 65
4.5.04 NumCachedSamplesValue State object 66
4.5.05 FlagState State object 67
4.5.06 CounterValue State object 67
4.5.07 GpioPinState State object 68
4.5.08 TimeOfDay State object 69
4.5.09 DayOfTheWeek State object 70
4.5.10 DayOfTheMonth State object 71
4.5.11 CurrentMonth State object 72
4.5.12 CurrentYear State object 73
4.6 Rule Objects 74
4.6.01 UserRule Rule object (Main) 74
4.6.02 UserRule Rule object (State Check) 75

4.6.03 UserRule Rule object (Triggered Actions) 76

5. Example Rulesets 77

5.1 Simple Data Forwarding 78

5.2 Timers and Conditionals 83

5.3 Email and SMS Alerts 90

5.4 Logging, Flags, and GPIO's 101

5.5 Counters, Averagers, Cache, and Custom Report packets 114

Addendum A: MQTT Extended Options Files (.mcfg) 128

A.01 Introduction 128

A.02 General Connection Settings 129

A.03 Last Will Connection Settings 129

A.04 SSL/TLS Connection Settings 130

A.05 JSON Filter-Format Files 130

A.06 External Control Via User Defined Events 131

Addendum B: JSON Filter-Format Files (.jff) 132

B.01 Introduction 132

B.02 Pattern Match Specifiers 132

B.03 Data Label Specifiers 132

B.04 Output Format Specifier 133

Addendum C: SoftHub Object Tables 134

C.01 Sensor Stream Object(s) Table 134

C.02 User Report Object(s) Table 134

C.03 Event Object(s) Table 134

C.04 Action Object(s) Table 135

C.05 State Object(s) Table 137

C.06 User Rule Object(s) Table 137

1. Introduction

This document provides an in-depth view of the SoftHub Configuration Utility. It will detail how to use the SoftHub Configuration Utility to create, edit, and deploy the file used to configure the SoftHub Application. While it will also provide a brief review of the SoftHub Platform itself, it is assumed that the reader has previously familiarized themselves with the companion document titled *Introduction: The OmnIoT SoftHub Platform*.

1.1 SoftHub Platform Components

Currently the OmnIoT SoftHub Platform consists of three primary components:

The SoftHub Application – This code runs on the physical hub device itself. On application startup, it reads a user created configuration file which the SoftHub's internal rule engine will continually execute. Users identify *Events* they are interested in and *Rules* to be evaluated as these events occur. Rules in turn consist of optional conditional *State* objects to be evaluated and *Actions* to be carried if the state logic is fully satisfied.

The SoftHub Configuration Utility – This is a Windows' PC application the user runs to build the configuration file that is to be read/executed by the SoftHub Application. In general there are two main sections in the rule file. The *System Options* section allows the user to set system-wide settings while the *Rule Engine* section allows the user to define the objects and rules to be executed by the rule engine itself.

The Remote Packet Capture Application – For applications that require data to be forwarded to one or more central servers, this application runs on the remote packet capturing server(s) which receive and decode incoming packets from one or more hub devices running the SoftHub Application. Currently the Remote Packet Capture Application runs as a Windows Service (though a similar Linux daemon is planned in the future). As packets are received and decoded, an external DLL is called which will be passed both the raw binary packet data as well as an optional fully decoded JSON or XML version of each packet. Users provide the external DLL and may then post-process the packets as needed. Both C# and C++ sample projects are provided for building this DLL.

1.2 SoftHub Rule Engine Mechanics

Architecturally, the SoftHub Application exists a set of “loosely coupled” subcomponents tied together and orchestrated by one central subcomponent, the SoftHub “Rule Engine”. A user defined ruleset will contain a group of objects which the Rule Engine will instantiate at startup. These objects can be looked at as individual stand-alone executables. As events are detected the rule engine will process any associated *Rule Objects* which in turn may evaluate a list of S*tate Objects*, which depending on their outcome may trigger the execution of one or more *Action Objects*. Each of these Action Objects may then interact with any of the SoftHub’s subcomponents to carry out the user specified action.

By way of an example a user may wish to connect to a set of BLE sensors every day at a specific time. The user would start by creating a “Time of Day Timer” action object. When executed, this object would register a timer with the SoftHub’s *Timer Management* subcomponent. At the designated time the Timer Management subcomponent would generate the associated *TimerExpired* event which would be received by the SoftHub’s Rule Engine subcomponent. The Rule Engine would then evaluate any rules the user had associated with this event. In this example the user would also create a *ConnectSensor* action object specifying the desired BLE sensor. That object would in turn send a message to the SoftHub’s *BLE Comms Manager* subcomponent requesting that the specified sensor be connected.

In the simple example above we have a few user created action objects (the time of day “timer” object and the “connect sensor” object) directing a couple of the SoftHub’s subcomponents (the Timer Management and BLE Comms Manager subcomponents). Lastly, we have the Rule Engine subcomponent at the heart of everything creating the objects and orchestrating their execution.

As described above, the heart of the OmnIoT SoftHub platform is the SoftHub Rule Engine. Users are provided a palette of objects which in turn are used to create a *ruleset*. Rulesets are the part of the configuration file that ultimately controls all aspects of how the SoftHub Application will behave. Rulesets can be thought of as *the list of things you want the hub device to do*, *and when and under what conditions you want those things done*. Currently there are six categories of SoftHub objects - *Sensor Stream* objects, *User Report* objects, *Event* objects, *State* objects, *Action* objects, and *User Rule* objects. In order to effectively configure the SoftHub’s rule engine it is vital to understand how these objects relate to each other and these four basic concepts:

(1) Any SoftHub subcomponent may detect or generate an event. When a specific event occurs an Event Object will be put into the Rule Engine’s event queue by the originating subcomponent.
(2) When the Rule Engine pops an Event Object from its event queue it will then evaluate one or more Rule Objects that the user has associated with the detected event.
(3) Rules are evaluated in the order in which the user has defined them. The *evaluation process* refers to the executing of any optional *logical conditional* statements the user has defined in relation to the rule. Logical conditionals are made up of one or more logically connected State Objects the user may have defined.
(4) If the optional conditional statement resolves to a Boolean “true” then all Action Objects associated with the rule will be executed (again in the order they are defined by the user).

Understanding the above four tenants is critical to effectively configuring the SoftHub's rule engine. The statements above touch on four of the six current rule engine object types. Below is a brief description of each object type -

Event Objects – Event objects may be put into the Rule Engine's "event queue" by any SoftHub subcomponent. Users may associate one or more Rule objects to be evaluated when the Rule Engine receives a specific event type. Every Rule object is triggered by exactly one Event object. In contrast, most Event objects are reoccurring and may trigger the evaluation multiple Rule objects.

Action Objects – Individual Action objects each describe a singular action to be carried when a Rule object's conditional statement has been evaluated as "true". Every Rule object will reference one or more Action objects.

State Objects – State objects allow the user to effectively associate logical conditional statements with the execution of a Rule object. Each State object will evaluate some aspect of the current system state as either "true" or "false". These objects in turn can be strung together via logical AND, OR, and NOT statements to create more complex logic as required.

Rule Objects – Rule objects are actually a collection of references to (1) a single triggering Event object, (2) one or more logically connected State objects, and (3) one or more Action objects. As previously described state objects are optional and are used to create conditional logic to determine whether the rule's action object list should be executed. The list of Action objects describe one or more actions to be performed when the conditional state logic has been fully evaluated and resolved to "true".

Sensor Stream Objects – Each of these objects will define an individual sensor that you want the SoftHub to connect to. Sensor Stream objects may either be for a specific type of sensor (e.g. *any* "BLE TI SensorTag sensor") or a specific physical sensor (e.g. *the* "BLE TI SensorTag sensor with the MAC ID of 9F77A834E722"). Note that Sensor Stream objects don't "do" anything on their own but rather are used as references in other objects.

User Report Objects – User Report objects simply allow the user to define their own data packets, in either binary, JSON, or XML formats, aggregating data from a variety of sources. These packets in turn can be forwarded to a remote server or logged to internal storage on the SoftHub device itself. As with Sensor Stream objects, User Report objects don't "do" anything on their own but rather are used as references in other objects.

One common attribute that all objects share is that the user must provide a unique *Object Name* when the object is initially created. The object name is the first parameter to be defined for all objects regardless of type. As will become apparent, it is important to name your objects logically as many objects will cross reference and build upon other objects.

1.3 Configuration File Contents and Deployment

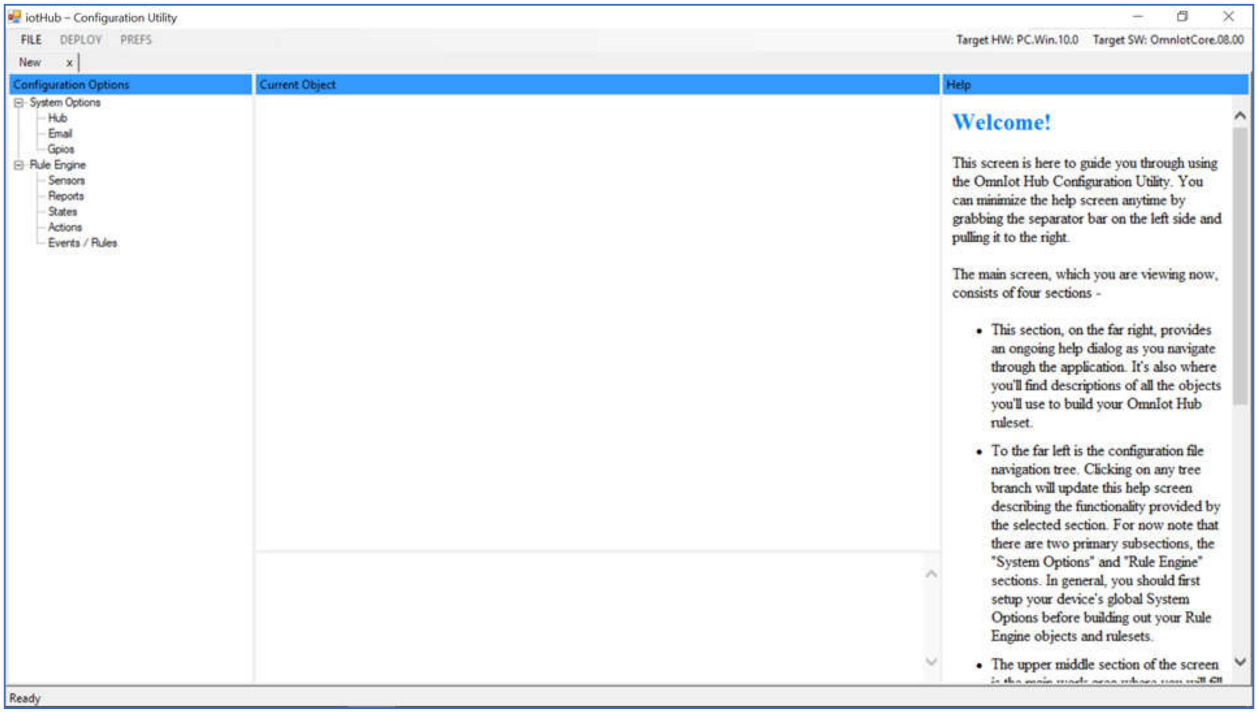
To configure the SoftHub Application, the user must create and place the “RuleEngine.xml” file on the physical hub hardware on which the SoftHub application is running. The exact location is specific to the hardware platform being used and is documented in the software installation package. But in general, the process is (1) the user runs the SoftHub Configuration Utility and generates the RuleEngine.xml file, (2) the user places the file in the appropriate directory on the hub device, and (3) the user restarts the SoftHub daemon or service on the hub device. At that point the SoftHub Application will read the configuration file and continuously execute its ruleset in the background.

As is apparent by the filename, the configuration file itself is a simple XML file. The file contains two main sections, the *System Options* settings which set system-wide global runtime options, and the *Rule Engine Directives* which describe all the rule engine objects referenced by the rule engine ruleset and the individual rule engine rules themselves. As will be apparent in coming sections, the UI of the SoftHub Configuration Utility closely mimics the contents of the configuration file.

NOTE: The RuleEngine.xml file should <u>NEVER</u> be edited by hand. It is assumed that the configuration file is syntactically correct when read by either the SoftHub Configuration Utility (for editing) or the SoftHub Application itself. Minimal error detection is performed in either application so invalid configuration files will have undefined results.

2. Using the SoftHub Configuration Utility

The SoftHub Configuration Utility is currently a somewhat minimal tool and as such has only one main screen as its UI.



Welcome!

This screen is here to guide you through using the OmnIot Hub Configuration Utility. You can minimize the help screen anytime by grabbing the separator bar on the left side and pulling it to the right.

The main screen, which you are viewing now, consists of four sections -

- This section, on the far right, provides an ongoing help dialog as you navigate through the application. It's also where you'll find descriptions of all the objects you'll use to build your OmnIot Hub ruleset.
- To the far left is the configuration file navigation tree. Clicking on any tree branch will update this help screen describing the functionality provided by the selected section. For now note that there are two primary subsections, the "System Options" and "Rule Engine" sections. In general, you should first setup your device's global System Options before building out your Rule Engine objects and rulesets.
- The upper middle section of the screen

Figure 1: The SoftHub Configuration Utility main screen

There are four main areas of interest. First and foremost, on the left side of the screen is a panel containing the configuration object navigation tree. To the far right is a collapsible context sensitive help screen panel. The main panel (upper center) is where the currently selected object in the navigation tree will be made available for editing. And at the bottom center is a panel where a running dialog of object cross references for the currently selected object is presented.

The sections below describe the tool's various components at a high level before delving into the actual objects and settings themselves.

2.1 The Toolbar Menu

When opening the Configuration Utility the user will first either open an existing file for editing or create a new file by using the “FILE” toolbar entry in the upper left hand portion of the application screen (figure 2). Once editing is complete, use “SAVE” to save the file with its current name or “SAVE AS” to rename the file before saving. Note that before deploying the configuration file to the actual hub device itself the file must be renamed “RuleEngine.xml”. However when working with several different configurations it is generally easier to give each file a descriptive name and rename it only when deploying it to the hub device.

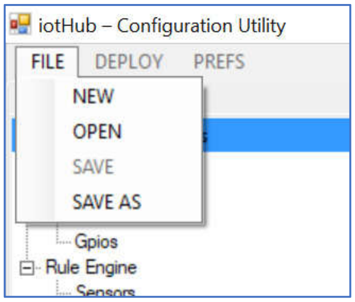
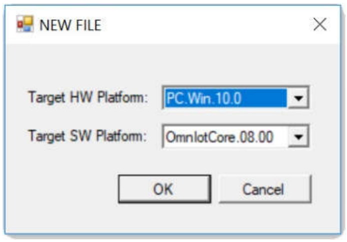


Figure 2: The Configuration Utility Toolbar

2.2 Setting the Target Platform

Configuration files are built to be backwards compatible however not all hub platforms will support the same functionality (e.g. Windows tablets will not support GPIO’s). For this reason, each configuration file is associated with a specific hardware and software target environment.

The user sets the target environments by filling in the appropriate information in the pop-up that will appear when “New” is selected from the FILE toolbar menu. Figure 3 shows the pop-up and its fields. Choosing the correct hardware target should be fairly obvious. When choosing the software target it is important that you know what version is running in the targeted hub device. Trying to load a file targeted for an incorrect platform into the SoftHub Application will result in the application logging an error and terminating.



Target HW Platform: PC.Win.10.0
Target SW Platform: OmnIotCore.08.00

Figure 3: Target Platform Pop-up

2.3 The Configuration Options Panel

The Configuration Options Panel runs down the left side of the screen and provides the navigation tree within the configuration file itself. Again, the tree closely mimic's the configuration file's contents at the highest two subsection levels. Under the "System Options" section, global system-wide runtime options are listed and can be selected for editing. Under the "Rule Engine" section users define the various building block objects to be created by the SoftHub rule engine as well as the Rule Objects themselves that dictate how the SoftHub Application will operate.

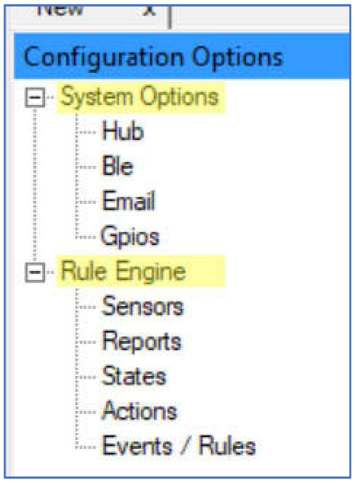


Figure 4: Highest level Configuration Tree nodes

Under the System Options tree are four sub-categories of options the user can edit. Left clicking on the "Hub" label will populate the Current Object pane with a set of general hub options that the user can edit and make any desired changes. Left clicking the "Ble" label will allow the user to edit the SoftHub's Bluetooth Low Energy interface options. Similarly, left clicking on the "Email" label will allow the user to enter email SMTP configuration information to enable the sending of email and SMS messages directly from the hub device. Lastly, left clicking on the "GPIOs" label will allow the user to define any input or output GPIO's they wish to manipulate via their ruleset.

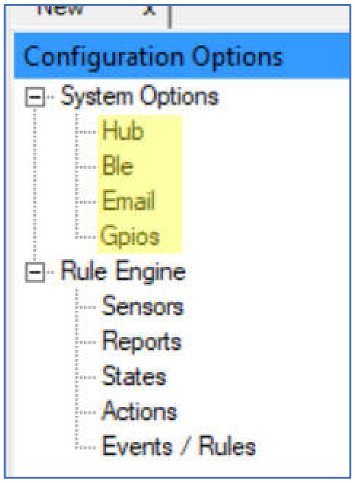
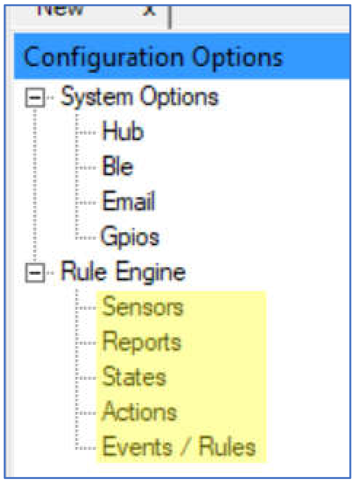


Figure 5: System Option Labels

Beneath the Rule Engine tree node are the various categories of objects used by the SoftHub's rule engine. As objects are created they will appear under their respective category node. Right clicking on any of the nodes and selecting "New" will present a pop-up allowing the user to pick the specific object that they wish to create. Once a specific object is selected the Current Object pane will be populated with all the parameters that define the selected object type. Once the object has been edited to the users' satisfaction, the object is saved and it will now appear as a node under the original object type branch.

The slight exception to this behavior is the "Events / Rules" category. Most Event objects are created as a side effect of creating some other object (excluding the AppStarting and AppStopping event objects). Since every rule object is triggered by exactly one event object, rules are created under the Event object nodes that populate under the "Events / Rules" node of the tree. As event nodes are added you can right click on those objects and select "New" to create a new Rule Object.

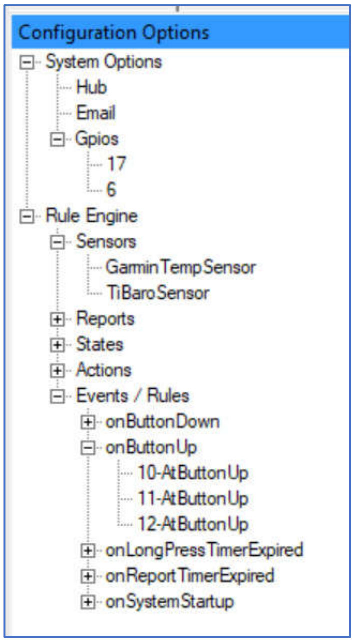


Configuration Options
- System Options
  - Hub
  - Ble
  - Email
  - Gpios
- Rule Engine
  - Sensors
  - Reports
  - States
  - Actions
  - Events / Rules

Figure 6: Rule Engine Object Type Labels

The example labeled "Figure 7" shows a typical tree with some nodes expanded. You can see two GPIO's that have been enabled, and two Sensor Stream objects as well. There are several Event object nodes shown with the "onButtonUp" event node object expanded to display its list of associated Rule objects. Left clicking on any lowest level object (e.g. the "TiBaroSensor" node) will populate the Current Object Pane with the values currently defining that object. Right clicking on an objects label will provide the user with two additional options, "Delete" or "Duplicate". Delete will delete the object while Duplicate will create a duplicate of the object with the same values pre-populated with the exception of the object name (which must be unique).

One important point to understand is that object nodes are created and listed alphabetically. This has no practical effect except in relation to Rule Objects. When an event is detected the rules are evaluated in the order listed. This may be important as one Rule object's Action objects may impact a State object referenced by a subsequent Rule object. Using numbers to prefix the rule object names (as in this example) allows the user to ensure proper execution order.



Configuration Options
- System Options
  - Hub
  - Email
  - Gpios
    - 17
    - 6
- Rule Engine
  - Sensors
    - GarminTempSensor
    - TiBaroSensor
  - Reports
  - States
  - Actions
  - Events / Rules
    - onButtonDown
    - onButtonUp
      - 10-AtButtonUp
      - 11-AtButtonUp
      - 12-AtButtonUp
    - onLongPressTimerExpired
    - onReportTimerExpired
    - onSystemStartup

Figure 7: A typical Configuration File navigation tree

As you navigate the Configuration Options panel its operation should become fairly apparent. Always refer to the Help Panel, detailed in the following section, for the full list of options on any given screen.

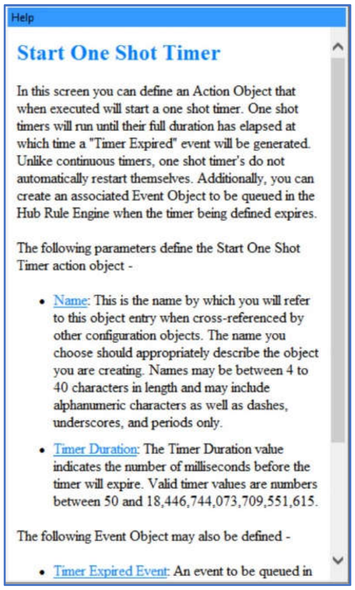
2.4 The Help Panel

Figure 8 illustrates an example of a typical context sensitive help screen. In this example a *OneShotTimer* action object would be selected in the Configuration Options object tree and the specific object's current settings would be displayed in the Current Object Panel.

If a higher-level object category node is currently selected (i.e. *not* an actual object node), the help file will give a description of the category as well as what mouse button menus are available.

If an actual object node is selected, the help screen provides information on the object and each of its editable parameters. Additionally, valid parameter value ranges would be provided where appropriate.

Note that this panel is also fully collapsible to provide additional screen space for the other panels if desired.



Start One Shot Timer

In this screen you can define an Action Object that when executed will start a one shot timer. One shot timers will run until their full duration has elapsed at which time a "Timer Expired" event will be generated. Unlike continuous timers, one shot timer's do not automatically restart themselves. Additionally, you can create an associated Event Object to be queued in the Hub Rule Engine when the timer being defined expires.

The following parameters define the Start One Shot Timer action object -

- Name: This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating. Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
- Timer Duration: The Timer Duration value indicates the number of milliseconds before the timer will expire. Valid timer values are numbers between 50 and 18,446,744,073,709,551,615.

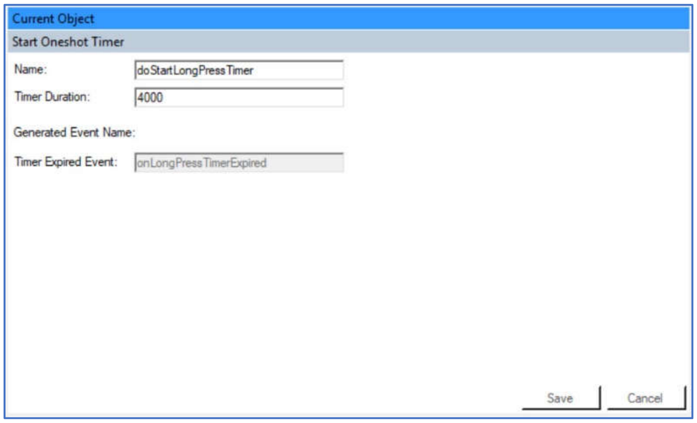
The following Event Object may also be defined -

- Timer Expired Event: An event to be queued in

Figure 8: A typical Help screen

2.5 The Current Object Panel

At the heart of the Configuration Utility main screen is the *Current Object* panel. This panel is where all system options are edited and all rule engine objects are created and/or edited. Figure 9 shows a typical object being edited in the Current Object panel.



Start Oneshot Timer

Name: doStartLongPressTimer

Timer Duration: 4000

Generated Event Name:

Timer Expired Event: onLongPressTimerExpired

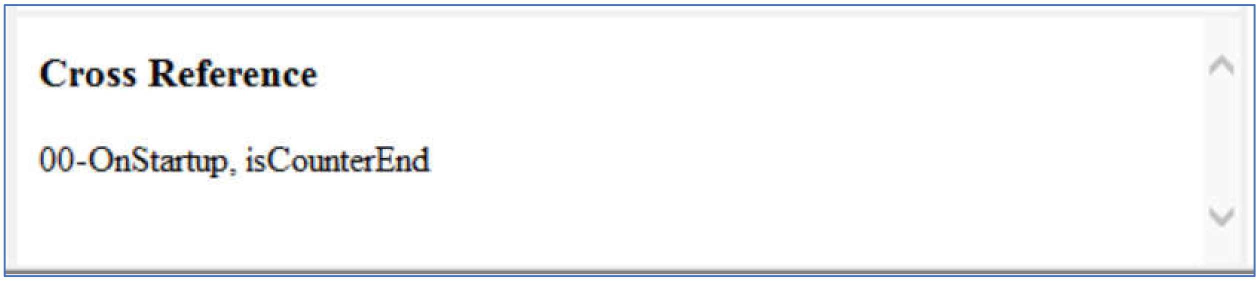
Figure 9: Typical object being edited in the Current Object panel

In the example the user has selected a *StartOneShotTimer* action object in the configuration object tree. It has been given an object name of “doStartLongPressTimer” and has a duration of 4000 milliseconds. An event object, “onLongPressTimerExpired”, will be created as well and will be queued up every time the “doStartLongPressTimer” timer expires.

While in this screen the user may make changes to any enabled parameters and save them via the “Save” button. Occasionally fields may be disabled due to cross references by other objects. For those fields, reference the Help screen for a full description of why the field is inactive and how to reactivate it if required.

2.6 The Object Info Panel

The *Object Information* panel, at the center bottom of the main screen, gives the user information regarding the object currently displayed in the Current Object edit panel. Due to the frequent cross references between objects you may be unable to delete an object when it would invalidate a referring object. This panel provides the user with a list of all objects cross-referencing the object currently being edited.



Cross Reference

00-OnStartup, isCounterEnd

Figure 10: Example Object Info cross references

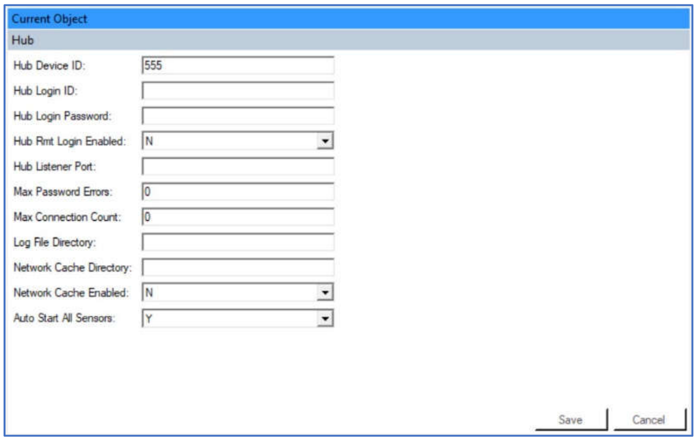
3. SoftHub System Options

The System Options section of the configuration tree is where global system-wide settings are made available for editing. The SoftHub's system options are broken into four categories, (1) general hub settings, (2) Bluetooth Low Energy interface settings, (3) email settings for optionally setting up an SMTP server, and (4) GPIO settings for enabling any input and output GPIO pins on the physical hub device. Left clicking on any of the sub-nodes (Hub, Ble, Email, or GPIOs) will populate Current Object pane with that categories' set of editable parameters. The sections below describe the settings currently available in each of the System Options categories, including their function and value ranges. Note that much of this information is also available directly in the Configuration Utility itself via the context sensitive Help panels.

3.1 Hub Options

In this section of the System Configuration options you will find a collection of global options which control various *general* components of the OmnIot SoftHub.

The table below describes the available options and their valid value ranges.



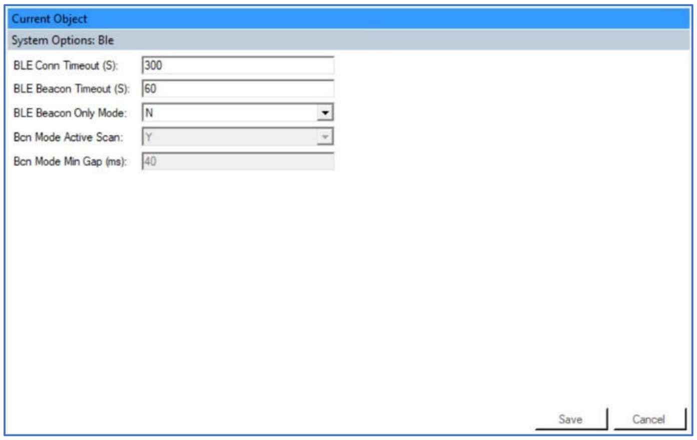
Hub Device ID: 555
Hub Login ID:
Hub Login Password:
Hub Rmt Login Enabled: N
Hub Listener Port:
Max Password Errors: 0
Max Connection Count: 0
Log File Directory:
Network Cache Directory:
Network Cache Enabled: N
Auto Start All Sensors: Y

Figure 11: Hub options screen

Field Name	Description	Valid Values
Hub Device ID	The Hub Device ID is a numeric value which is used to uniquely identify each hub you deploy. This identity value is particularly relevant when sending data packets from multiple hub devices to a central server in the cloud.	A numeric value between 0 and 65,535.
Hub Login ID	The optional Hub Login ID specifies the login ID credential to be required if remote logging in to the hub is enabled.	Valid values for the Hub Login ID are alphanumeric strings of length between 4 and 24 characters, which may include dashes, underscores, and dots.
Hub Login Password	The optional Hub Login Password specifies the login password credential to be required if remote login is enabled.	Valid values for the Hub Login Password are alphanumeric strings of length between 4 and 24 characters, which may include dashes, underscores, and dots.
Hub Remote Login Enabled	The Hub Remote Login Enabled option determines whether remote logging into the hub device is allowed.	Y or N
Hub Listener Port	This value indicates what listener port the SoftHub Application will use for remote login capability (if enabled).	Value ranges are from 1 to 65,535, however the user should ensure that the chosen port is not currently in use.
Max Password Errors	The Max Password Errors field determines the maximum number of password errors to be allowed before a remote connection will be terminated.	Valid values for the Max Password Errors field are numerics from 0 to 100 where 0 indicates no limit on password errors.

Max Connection Count	The Max Connection Count field determines the maximum number of concurrent login connections allowed when the Hub Remote Login option is enable.	Valid values for the Max Connection Count field range from 0 to 16.
Log File Directory	The Log File Directory option determines where the hub writes all log files in the local disk directory. This includes the system log file as well as any user generated report or packet log files.	The Log File Directory option must be in a valid directory format for the target platform.
Network Cache Directory	The Network Cache Directory option determines where the hub writes all network cache files in the local disk directory when network caching is enable.	The Network Cache Directory option must be in a valid directory format for the target platform.
Network Cache Enabled	The Network Cache Enabled option determines whether network caching will be available when the system has lost its network connection. Network caching allows the system to save packets that are undeliverable to local disk space. Once a connection has been reestablished the undelivered packets will be forwarded to the originally targeted remote server.	Y or N
Auto Start All Sensors	The Auto Start All Sensors option determines whether the sensors defined in the SoftHub rule set are to be automatically started at application startup. When this option is set to "NO", the user must manually enable each sensor stream by executing an *EnableSensorStream* action object when appropriate.	Y or N

3.2 Ble Settings



Current Object

System Options: Ble

BLE Conn Timeout (S): 300
BLE Beacon Timeout (S): 60
BLE Beacon Only Mode: N
Bcn Mode Active Scan: Y
Bcn Mode Min Gap (ms): 40

Figure 12: Ble Options screen

The Ble System Options settings allow the user to set basic behaviors of the SoftHub's Bluetooth Low Energy interface settings. Supported Bluetooth Low Energy sensors are of two types, sensors that provide a traditional peer to peer connection, or "beacon" sensors which publicly advertise their data to be read by any listening hub device. The SoftHub provides two modes the Ble interface can be run in, a "mixed mode" which supports both types of sensors simultaneously, and a "beacons only" mode which is tailored explicitly to applications utilizing beacon sensors only. In practice, if you're application is beacon only it is better to use the beacon only mode. This is because, when mixing connection types, depending on the beacon sensor type there may be significant packet loss and/or connectivity issues. See the notes on the individual fields below for more information.

The table below describes the available options and their valid value ranges.

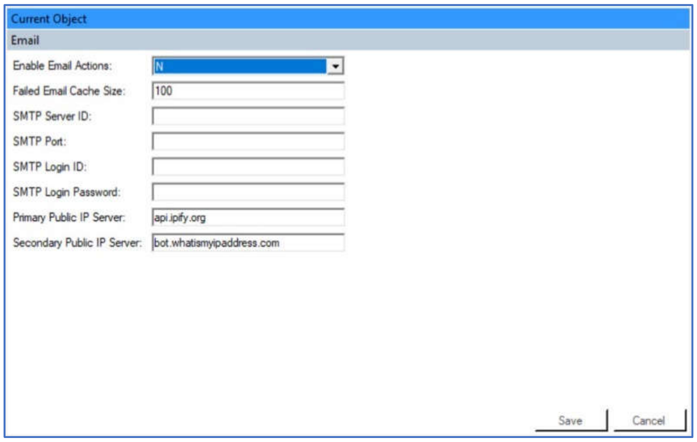
Field Name	**Description**	**Valid Values**
BLE Conn Timeout (S)	The BLE Conn Timeout is a numeric which indicates the amount of time, in seconds, that a connected Bluetooth Low Energy sensor can be inactive before being considered disconnected. Note that this value has no significance when "BLE Beacon Only Mode" is set to "Yes".	Valid values for the BLE Conn Timeout are integers between 0 and 65,535.
BLE Beacon Timeout (S)	The BLE Conn Timeout is a numeric which indicates the amount of time, in seconds, that a connected Bluetooth Low Energy beacon type sensor can be inactive before being considered disconnected. Note that in this context, "connected" refers to a beacon being actively monitored though not connected in a traditional Ble sense.	Valid values for the BLE Beacon Timeout are integers between 0 and 65,535, where 0 indicates a beacon once connected will never time out.
BLE Beacon Only Mode	Enabling BLE Beacon Only Mode tells the SoftHub to use the lower level Linux API's to monitor for beacon packets. This mode allows for unfiltered access to beacon packets however it is incompatible with normal (non-beacon) BLE connections. Enable this option only if you will not be connecting any BLE sensors that are not beacon sensor. Note that when attempting to connect a non-beacon Ble sensor in this more you will receive many "connection failed" messages in the SoftHub log.	This values are “Y” or “N”.

Bcn Mode Active Scan	Enabling the Beacon Mode Active Scan allows the system to connect with beacons by name (type) vs mac address. The cost is that the additional overhead of continually scanning devices for their device name will negatively impact the battery life of all nearby beacons detected. This may be acceptable if beacons are moving in an out or range transiently, but when you expect to use a small set number of sensors in your application it is preferable to specify the sensor's mac addresses explicitly and disable this option. Note that this value has no significance when "BLE Beacon Only Mode" is set to "No".	This values are “Y” or “N”.
Bcn Mode Min Gap (ms)	The BLE Conn Timeout is a numeric value that indicates a minimal gap, in milliseconds, between two packets being reported when in Beacon Only Mode. Since each packet may be received on up to three separate channels, this setting allows the user to filter out possible duplicate packets being reported. Note that this value has no significance when "BLE Beacon Only Mode" is set to "No".	Valid values for the Bcn Mode Min Gap are integers between 0 and 65,535.

3.3 Email Settings

This section of the system configuration options provides a collection of options used to enable the SoftHub to directly send email and SMS messages via the *SendEmail* action object.

The table below describes the available options and their valid value ranges.



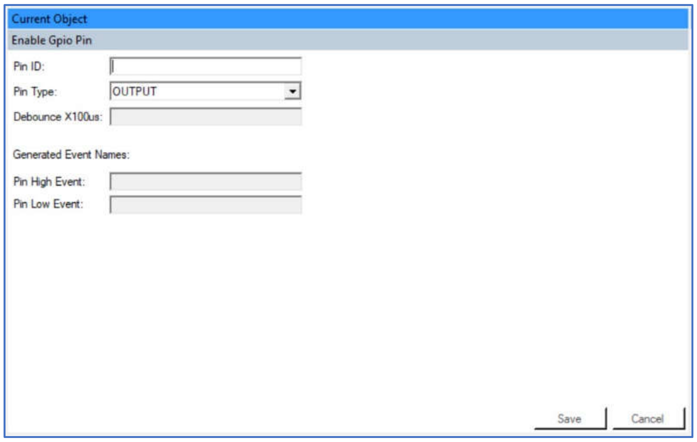
Enable Email Actions: N
Failed Email Cache Size: 100
SMTP Server ID:
SMTP Port:
SMTP Login ID:
SMTP Login Password:
Primary Public IP Server: api.ipify.org
Secondary Public IP Server: bot.whatismyipaddress.com

Figure 13: Email Options screen

Field Name	**Description**	**Valid Values**
Enable Email Actions	The Enable Email Actions option determines whether the OmnIot SoftHub has been configured to send remote email and SMS messages. Setting this option to "NO" will disable the *SendEmail* action object in the OmnIot Hub rule engine.	Y or N
Failed Email Cache Size	The Failed Email Cache Size option determines the maximum number of emails that will be cached when the OmnIot SoftHub is unable to successfully send emails due to a network connection being unavailable or some other failure.	Valid values for the failed email cache size are integers between 0 and 250.
SMTP Server ID	The SMTP Server ID option identifies the SMTP mail server that the OmnIot SoftHub should use when sending outgoing email or SMS messages.	This value must be in a valid DNS name format.
SMTP Port	The SMTP Port option identifies the SMTP port that the OmnIot SoftHub should use when sending outgoing or SMS messages to the target SMTP server.	Valid SMTP port values are integer values between 0 and 65535.
SMTP Login ID	The SMTP Login ID option identifies the login ID credential that the OmnIot SoftHub should specify when logging in to the remote SMTP server.	Limited to the character set of all alpha-numerics, plus any of the following "!@#$%&-_. =".
SMTP Login Password	The SMTP Login Password option identifies the login ID password that the OmnIot SoftHub should specify when logging into the remote SMTP server.	Limited to the character set of all alpha-numerics, plus any of the following "!@#$%&-_.= ".
Primary Public IP	The Primary Public IP Server identifies a remote server that provides a service which will return the callers originating IP address.	This value must be in a valid DNS name format.
Secondary Public IP Server	The Secondary Public IP Server identifies a remote server that provides a service which will return the callers originating IP address. This server will be used only when the Primary Public IP Server is unable to be reached.	This value must be in a valid DNS name format.

3.4 GPIO Settings

The Enable GPIO Pin screen allows the user to define the set of GPIO pins to be referenced in the SoftHub ruleset. Additionally, when defining *an input pin*, users can create associated Event objects to be queued for the SoftHub’s rule engine whenever the pin transitions from one state (i.e. "high" or "low") to the other. Similarly, pins defined as *output pins* may be manipulated via the *SetGpioPinLow*, *SetGpioPinHigh*, and *ToggleGpioPin* Action objects.



Pin Type: OUTPUT

Figure 14: The Enable GPIO Pin options screen

The table below describes the available options and their valid value ranges.

Field Name	**Description**	**Valid Values**
Pin ID	The pin ID field allows a user to select the GPIO pin to be enabled.	Must be a valid pin number for the target hardware platform.
Pin Type	The Pin Type field allows the user to identify the selected GPIO pin as either an INPUT or OUTPUT pin. Note that only pins that are enabled as input pins will generate GPIO transition events in the OmnIot SoftHub's logic engine.	INPUT or OUTPUT
Debounce X100us	The Debounce X100us option allows the user to set a GPIO input pin's debounce time in increments of 100 microseconds. For instance, to set a debounce time of 5 milliseconds, set this option to 50.	Valid values for this field are integer numbers between 0 and 65535.
Pin High Event	When defining an input pin, this field names a GpioPinHigh Event object to be queued in the SoftHub's rule engine whenever the specified input pin transitions from a low to a high state.	Valid object names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Pin Low Event	When defining an input pin, this field names a GpioPinLow Event object to be queued in the SoftHub's rule engine whenever the specified input pin transitions from a high to a low state.	Valid object names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.

4. SoftHub Rule Engine Objects

This section will go through each of the object types and individual objects currently supported by the most recent SoftHub rule engine. Many rule engine objects are hierarchical and so may incorporate references to other objects in their definition. For this reason all object definitions require as their first parameter a user selected *Name* that it will be referenced by. Names should be chosen to clearly identify the object type and function in as much as possible.

Though it has been stated elsewhere in this document it is worth repeating that the central operating tenant of the SoftHub rule engine is –

(1) As supported events are detected by the various components of the SoftHub application, Event Objects are created and placed in the rule engine's event queue.
(2) As they arrive, the SoftHub's rule engine will pop an event object off its event queue and will sequentially evaluate each Rule Object the user has associated with the particular Even Object.
(3) Each Rule Object may have optional conditional logic associated with its execution. This state logic consists of one or more State Objects connected by a series of Boolean operators.
(4) If the conditional logic ultimately evaluates to "true" (or is omitted entirely), each of the Action Objects referenced by the Rule Object will in turn be executed sequentially.

Note that if you are running older versions of the SoftHub application on your hub devices, all listed objects may not be supported on the older platforms. Properly setting the target software platform when initially creating your configuration file will hide unsupported object types within the SoftHub Configuration Utility. This is normal and expected behavior.

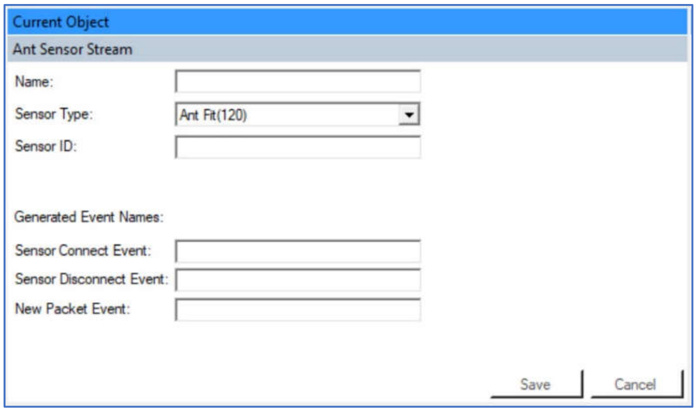
4.1 Sensor Stream Objects

Sensor Stream Objects describe the sensors that you want to make available to your hub device. The OmnIoT SoftHub Platform has an ever-growing library of supported sensors to choose from. Adding a new sensor to your application is a simple as choosing it from a dropdown menu and filling the required parameters to ensure you connect the proper sensor. In the SoftHub configuration file, sensors may be mixed and matched from multiple manufacturers or even different technologies (if supported by your hub hardware). Currently the SoftHub supports ANT and BLE sensor technologies, however future releases will be expanded to other include other low energy sensor technologies as well.

4.1.01 AntSensorStream Sensor Stream object

Creating a new *AntSensorStream* object allows you to choose an ANT sensor to connect to your hub. Here you pick the sensor type and optionally the specific sensor serial number. You may then reference this connection and any of the sensor data fields it provides in your Action, State, and Event, and Report objects.

The table below describes the available options and their valid value ranges.



Current Object
Ant Sensor Stream
Name:
Sensor Type: Ant Fit(120)
Sensor ID:

Generated Event Names:
Sensor Connect Event:
Sensor Disconnect Event:
New Packet Event:

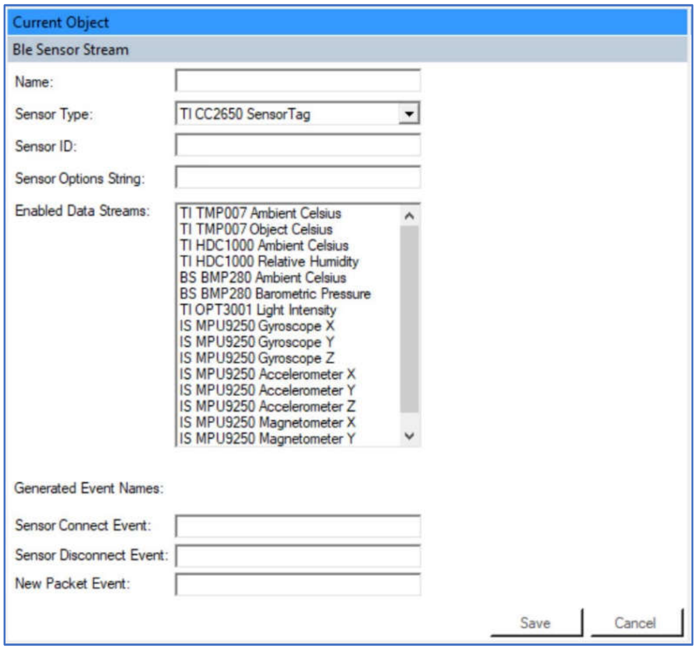
Figure 15: The Ant Sensor Stream object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Sensor Type	This is a drop-down list of ANT sensors supported for the currently selected target hardware and software platform. Use this drop-down list to select the type of Ant sensor you wish to connect to.	Any listed sensor type.
Sensor ID	This is an optional field where you can enter a specific sensor ID. If you do not specify a sensor ID the system will connect to the first available sensor it finds that matches the specified Sensor Type.	Valid Senor ID values are numeric values between 1 and 65,535.
Sensor Connect Event	This field names a *SensorConnect* Event object to be queued in the SoftHub's rule engine whenever a successful connection has been established with the sensor being defined.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Sensor Disconnect Event	This field names a *SensorDisconnect* Event object to be queued in the SoftHub's rule engine whenever a disconnection has been detected from the sensor being defined. (due to either a *DisableSensorStream* Action being triggered or an unplanned loss of signal).	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
New Packet Event	This field names a *NewSensorPacket* Event object to be queued in the Hub's Rule Engine whenever a new packet has been received from the sensor being defined.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.

4.1.02 BleSensorStream Sensor Stream object

Creating a new *BleSensorStream* object allows you to choose an BLE sensor to connect to your hub. Here you pick the sensor type and optionally the specific sensor mac address. You may then reference this connection and any of the sensor data fields it provides in your Action, State, and Event, and Report objects.

The table below describes the available options and their valid value ranges



Current Object

Ble Sensor Stream

Name:

Sensor Type: TI CC2650 SensorTag

Sensor ID:

Sensor Options String:

Enabled Data Streams:

TI TMP007 Ambient Celsius
TI TMP007 Object Celsius
TI HDC1000 Ambient Celsius
TI HDC1000 Relative Humidity
BS BMP280 Ambient Celsius
BS BMP280 Barometric Pressure
TI OPT3001 Light Intensity
IS MPU9250 Gyroscope X
IS MPU9250 Gyroscope Y
IS MPU9250 Gyroscope Z
IS MPU9250 Accelerometer X
IS MPU9250 Accelerometer Y
IS MPU9250 Accelerometer Z
IS MPU9250 Magnetometer X
IS MPU9250 Magnetometer Y

Generated Event Names:

Sensor Connect Event:

Sensor Disconnect Event:

New Packet Event:

Figure 16: The BLE Sensor Stream object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Sensor Type	This is a drop-down list of Bluetooth LE sensors supported for the currently selected target hardware and software platform. Use this drop-down to select the type of Bluetooth LE sensor you wish to connect to.	Any listed sensor type.
Sensor ID	This is an optional field where you can enter a specific sensor ID. If you do not specify a sensor ID the system will attach to the first available sensor it finds that matches the specified Sensor Type.	Valid Sensor ID values are 17 character strings in a valid BLE mac address format.
Sensor Options String	This is an optional field where you can specify sensor specific options such as a sensor name alias or pairing pin code. Options for individual sensors are detailed in the supported sensors spreadsheet provided as part of the installation package. Option strings must have the format of OPTION="????" where OPTION is the option specifier and ???? is the option value (which must be in quotes). More than one option can be passed by placing a single semicolon between options (spaces are not allowed).	A string consisting of one or more user options separated by a single semicolon (with no spaces). As an example, ALIAS=”acmeTemp”;”PAIRPIN=”000000” Would be a valid string specifying two user options, one an alias device name and the other a pairing pin.
Enabled Data Streams	Here you will find a list of the specific data streams supported by the currently chosen Sensor Type. Highlight and select the sensor data stream entries for each of the streams you wish to enable. Note that only data from the streams selected will be available the system.	Multi-select one or more sensor data streams to be enabled.
Sensor Connect Event	This field names a *SensorConnect* Event object to be queued in the SoftHub's rule engine whenever a successful connection has been established with the sensor being defined.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.

Sensor Disconnect Event	This field names a *SensorDisconnect* Event object to be queued in the SoftHub's rule engine whenever a disconnection has been detected from the sensor being defined. (due to either a *DisableSensorStream* Action being triggered or an unplanned loss of signal).	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
New Packet Event	This field names a *NewSensorPacket* Event object to be queued in the Hub's Rule Engine whenever a new packet has been received from the sensor being defined.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.

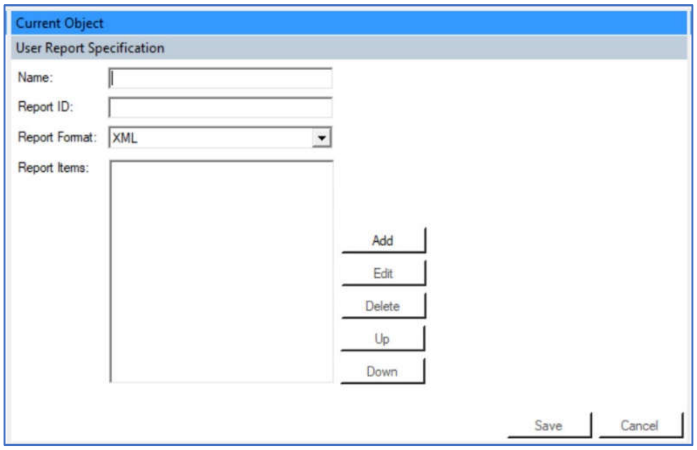
4.2 User Report Objects

Custom tailored report packets can be created via the *User Report Object*. This allows the user to aggregate data values from multiple sources and consolidate them into a single packet. The packets can then be sent to one or more remote servers or logged to the SoftHub's internal storage. Currently these packets may include the most recent data values from one or more connected sensors, averaged data values, cached data values, or momentary GPIO pin values. Packets may be generated in either binary, JSON, or XML format. User Reports with no items can be used to indicate user defined event detections.

4.2.01 UserReport User Report object (Main)

The *UserReport* object allows the user to create, name, and add report sections to a custom packet. These custom packets can then be sent to a remote cloud server or logged to the hub devices internal storage via the *SendReport* and *LogReport* action objects respectively.

The table below describes the available options and their valid value ranges.



Report Format: XML

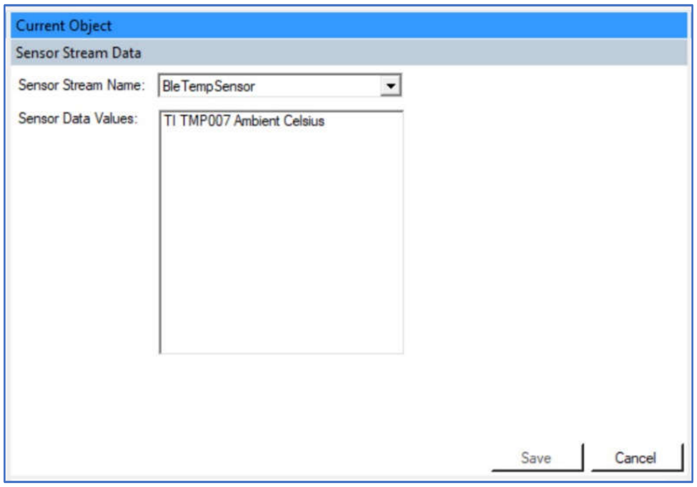
Figure 17: The User Report summary screen

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Report ID	The Report ID field gives the report a unique identification to indicate what fields the report packet contains. This is particularly important when generating binary report packets.	Report IDs may be alphanumeric for JSON or XML reports but must be unsigned 16-bit integers for binary reports.
Report Format	The Report Format field allows you choose whether the generated report packets will be in JSON, XML or binary format. Binary formatted report packets will be much smaller than JSON or XML formatted packets, while JSON/XML formatted packets have the benefit of being human eye readable and interchangeable with many other applications.	BIN, JSON, or XML
Report Items	In the Report Items list-box, select the data items that you wish to include in the report packet. Clicking the *Add* button will generate a pop-up with a drop-down of the different types of data items that can be added to your report. Each data item will in turn have its own data entry screen and fields. Currently supported Report Item types include Sensor Data Values, Averaged Sensor Data Values, Cached Sensor Data Values, and GPIO Pin Values. Note also that by using the *Edit*, *Delete*, *Up*, and *Down* buttons you are able to further customize the format of your report.	N/A

4.2.02 UserReport User Report object (Sensor Stream Data)

Clicking “Add” in the main User Report screen and choosing “Sensor Stream Data” will present the Sensor Stream Data selection screen as shown in Figure 17. When adding sensor data to your report you can limit the data you would like to include by selecting only the data values within the packet that are of interest. Choosing from the drop-down list of previously defined sensor streams, the user will see a list of available data values that stream provides.

The table below describes the available options and their valid value ranges.



Sensor Stream Name: BleTempSensor

Sensor Data Values: TI TMP007 Ambient Celsius

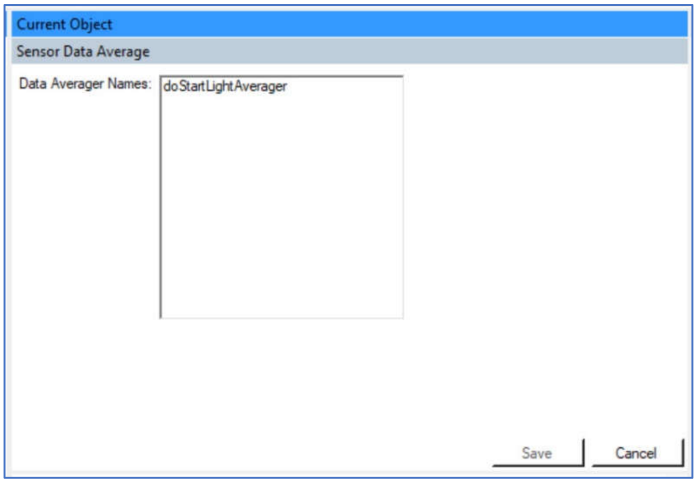
Figure 18: Add Sensor Stream Data screen

Field Name	**Description**	**Valid Values**
Sensor Stream Name	This drop-down will present a list of all previously defined *SensorStream* objects.	Any value in the list.
Sensor Data Values	This multi-select list will contain each of the data value fields previously enabled for the selected SensorStream object.	One or more values in the list.

4.2.03 UserReport User Report object (Sensor Data Average)

Clicking “Add” in the main User Report screen and choosing “Sensor Data Average” will present the Sensor Data Average selection screen as shown in Figure 18. This is a multi-select box, allowing the user to choose one or more previously defined data averager objects.

The table below describes the available options and their valid value ranges.



Data Averager Names: doStartLightAverager

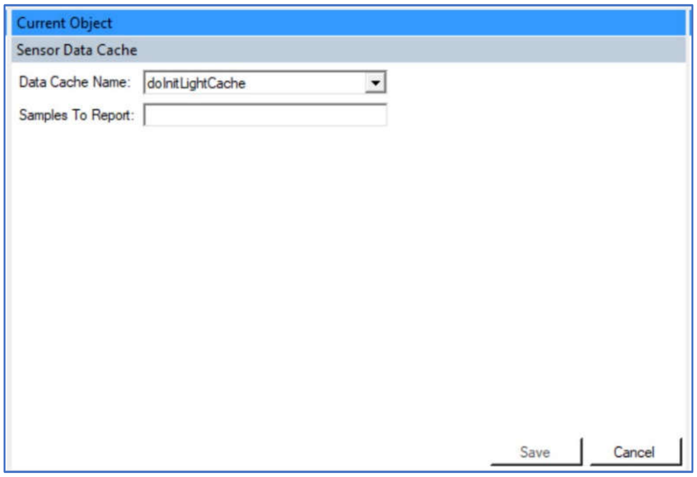
Figure 19: Add Sensor Data Average screen

Field Name	**Description**	**Valid Values**
Data Averager Names	This drop-down will provide a list of Sensor Data Averager objects created from previous definitions of any *InitSensorDataAverager* Action objects.	Any previously defined data averager object.

4.2.04 UserReport User Report object (Sensor Data Cache)

Clicking “Add” in the main User Report screen and choosing “Sensor Data Cache” will present the Sensor Data Cache selection screen as shown in Figure 19. Choosing a previously defined Sensor Data Cache object from the dropdown and selecting how many samples to report will include the most recent “N” samples that have been added to the selected cache.

The table below describes the available options and their valid value ranges.



Data Cache Name: doInitLightCache

Samples To Report:

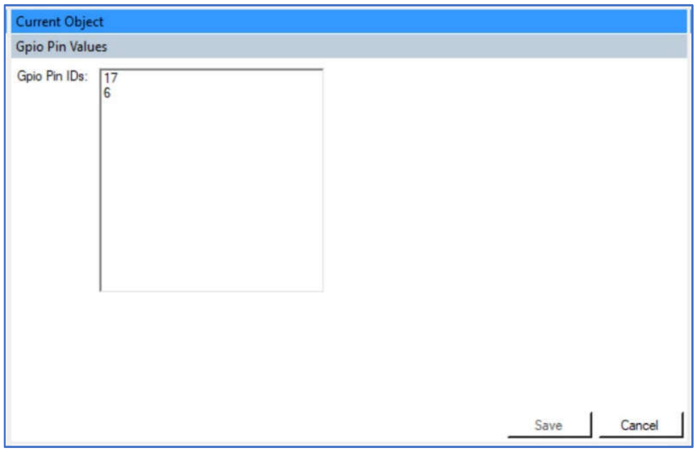
Figure 20: Add Sensor Data Cache screen

Field Name	**Description**	**Valid Values**
Data Cache Name	This drop-down will include a list of Sensor Data Cache objects created from previous definitions of any *InitSensorDataCache* Action objects.	Any previously defined Sensor Data Cache object.
Samples to Report	Here you can specify the number of cached values to be included in the report. The values reported will reflect the most recently cached values going back in time up until the number of requested values are reported or the number of currently cached values have been exhausted.	This value must be a number in the range of 1 to the defined cache size.

4.2.05 UserReport User Report object (GPIO Pin Values)

Clicking “Add” in the main User Report screen and choosing “GPIO Pin Values” will present the GPIO Pin Values selection screen as shown in Figure 20. Here the user can select one or more GPIO values to be included in their custom report packet. It is important to understand that the GPIO values are “momentary”, i.e. will reflect the GPIO value only at the specific moment the report was assembled.

The table below describes the available options and their valid value ranges.



Gpio Pin IDs:
17
6

Figure 21: Add GPIO Pin Values screen

Field Name	**Description**	**Valid Values**
GPIO Pin ID's	This is a multi-select list of GPIO pins in your system that have been previously enabled in the GPIOs System Options screen.	Multi-select one or more GPIO pin(s) from the list.

4.3 Event Objects

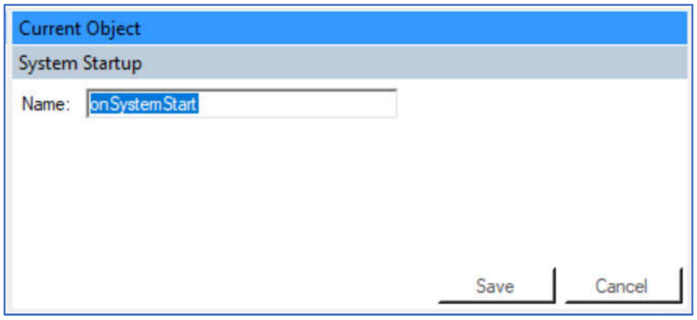
As events occur or are detected by different components of the SoftHub, a corresponding *Event Object* will be put into the rule engines' event queue. The events are then popped off the queue by the rule engine in "first in first out" order. If the user has defined any rules to be triggered by the specific event, then each rules' optional conditional logic will be evaluated and if true will result in the rule's action object list being executed. It is important to remember that both rules and action objects will be evaluated/executed in the order defined in the SoftHub Configuration File.

With the exception of the *AppStarting* and *AppStopping* event objects, all other event objects are created as a side effect of creating some other object. For instance, during the process of creating a new BleSensorStream object the user will have the opportunity to create associated event objects for when the sensor connects, disconnect, or sends a new packet.

4.3.01 AppStarting Event object

A *AppStarting* Event object is automatically placed as the first event object in the SoftHub's rule engine on application startup. Rules that are typically triggered at startup include rules to initializing flags, counters, connecting sensors, etc..

The table below describes the available options and their valid value ranges.



Name: onSystemStart

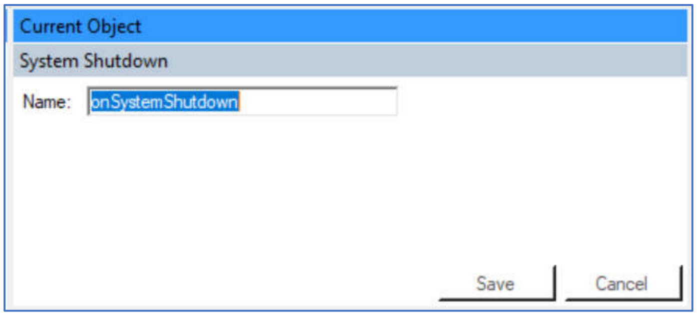
Figure 22: The System Startup Event object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.

4.3.02 AppStopping Event object

A *AppStopping* Event object will be placed in the SoftHub rule engine's event queue once a stop command has been received either externally or internally. Typically rules performing any system cleanup activities would be associated with the AppStopping event if required.

The table below describes the available options and their valid value ranges.



Name: onSystemShutdown

Figure 23: The System Shutdown Event object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.

4.3.03 UserDefinedEvent Event object

A *UserDefinedEvent* Event object will be placed in the SoftHub rule engine's event only when a message from a remote MQTT server has been received specifying the name associated with the event.

The table below describes the available options and their valid value ranges.

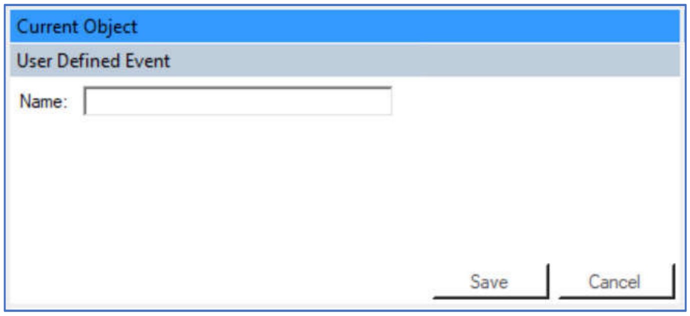


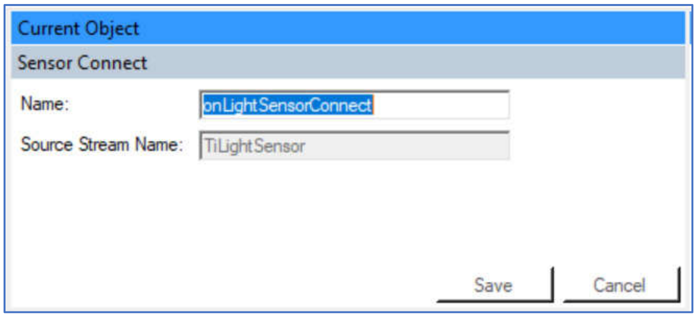
Figure 24: The User Defined Event object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when triggering it from a remote MQTT broker message. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.

4.3.04 SensorConnect Event object

SensorConnect Event objects are created as a side effect of defining a Sensor Stream object. During that process if the user so chooses they may elect to associate a SensorConnect event object to be queued when the Sensor Stream being created connects to the system.

The table below describes the available options and their valid value ranges.



Name: onLightSensorConnect

Source Stream Name: TiLightSensor

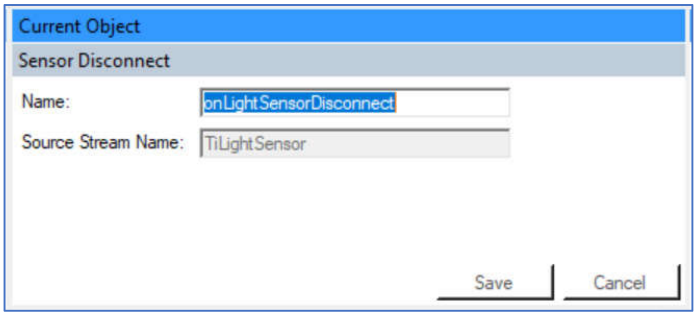
Figure 254: The Sensor Connect Event object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Sensor Stream Name	This field is prefilled and not editable. It names the *SensorStream* object that the user previously created and associated with this SensorConnect event.	N/A

4.3.05 SensorDisconnect Event object

SensorDisconnect events are created as a side effect of defining a Sensor Stream object. During that process if the user so chooses they may elect to associate a SensorDisconnect event object to be queued when the Sensor Stream being described disconnects from the system.

The table below describes the available options and their valid value ranges.



Name: onLightSensorDisconnect

Source Stream Name: TiLightSensor

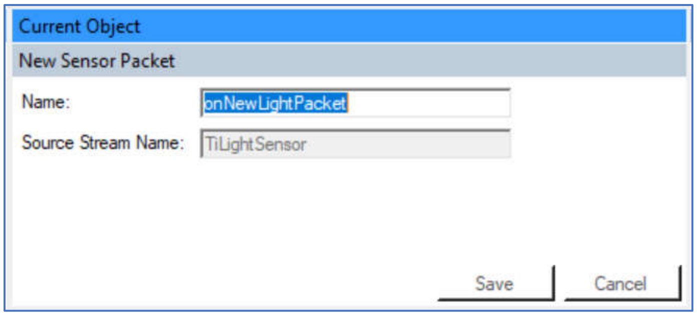
Figure 26: The Sensor Disconnect Event object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Sensor Stream Name	This field is prefilled and not editable. It names the *SensorStream* object that the user previously created and associated with this SensorDisconnect event.	N/A

4.3.06 NewSensorPacket Event object

NewSensorPacket events are created as a side effect of defining a Sensor Stream object. During that process if the user so chooses they may elect to associate a NewSensorPacket event object to be queued every time a new packet is received from the specified Sensor Stream.

The table below describes the available options and their valid value ranges.



Name: onNewLightPacket

Source Stream Name: TiLightSensor

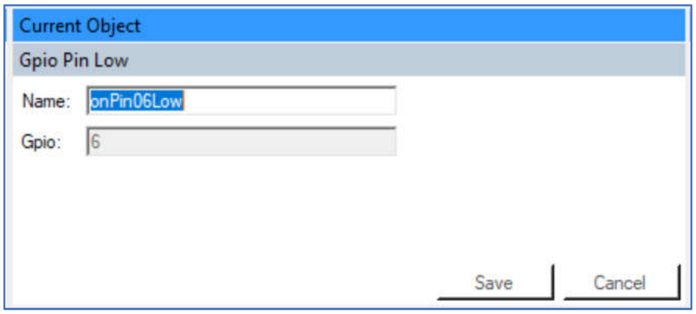
Figure 27: The New Sensor Packet Event object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Sensor Stream Name	This field is prefilled and not editable. It names the *SensorStream* object that the user previously created and associated with this SensorDisconnect event.	N/A

4.3.07 GpioPinLow Event object

GpioPinLow events are created as a side effect of enabling an input GPIO in the system options. During that process if the user so chooses they may elect to associate a GpioPinLow event to be queued every time the pin being defined transitions from a high to a low state.

The table below describes the available options and their valid value ranges.



Name: onPin06Low

Gpio: 6

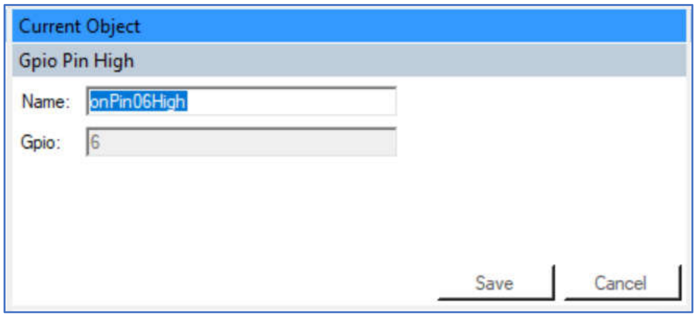
Figure 287: The GPIO Pin Low Event object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
GPIO	This field is prefilled and not editable. It names the input GPIO pin that the user previously created and associated with this GpioPinLow event.	N/A

4.3.08 GpioPinHigh Event object

GpioPinHigh events are created as a side effect of enabling an input GPIO in the system options. During that process if the user so chooses they may elect to associate a GpioPinHigh event to be queued every time the pin being defined transitions from a low to a high state.

The table below describes the available options and their valid value ranges.



Name: onPin06High
Gpio: 6

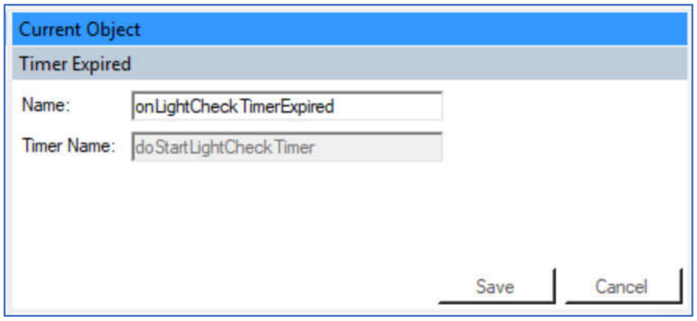
Figure 29: The GPIO Pin High Event object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
GPIO	This field is prefilled and not editable. It names the input GPIO pin that the user previously created and associated with this GpioPinHigh event.	N/A

4.3.09 TimerExpired Event object

TimerExpired objects are created as a side effect when the user defines a timer through any of the timer action object screens (e.g. *StartOneShotTimer* Action objects). During that process the user has the option to have a *TimerExpired* event queued anytime the timer being defined expires.

The table below describes the available options and their valid value ranges.



Name: onLightCheckTimerExpired
Timer Name: doStartLightCheckTimer

Figure 30: The Timer Expired Event object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Timer Name	This field is prefilled and not editable. It names the timer object that the user previously created and associated with this TimerExpired event.	N/A

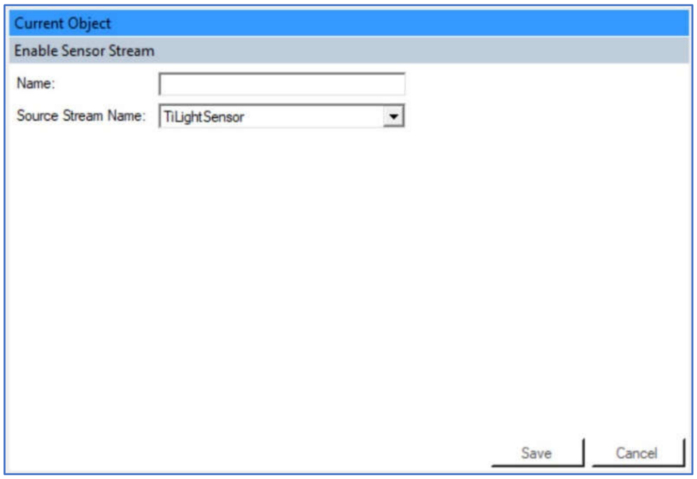
4.4 Action Objects

This section describes each of the currently available *Action Objects* for use with the SoftHub Application. Action objects should be thought of as simple stand-alone executable objects. When a Rule object's optional conditional logic is resolved as "true", its associated list of Action objects will be executed sequentially in the order they are listed.

4.4.01 EnableSensorStream Action object

When executed, the *EnableSensorStream* Action object will enable a previously defined Sensor Stream object. When enabled, the SoftHub application will continuously attempt to connect to the sensor specified by the associated Sensor Stream object. To later disable/disconnect the Sensor Stream, execute a subsequent *DisableSensorStream* Action object.

The table below describes the available options and their valid value ranges.



Name:

Source Stream Name: TiLightSensor

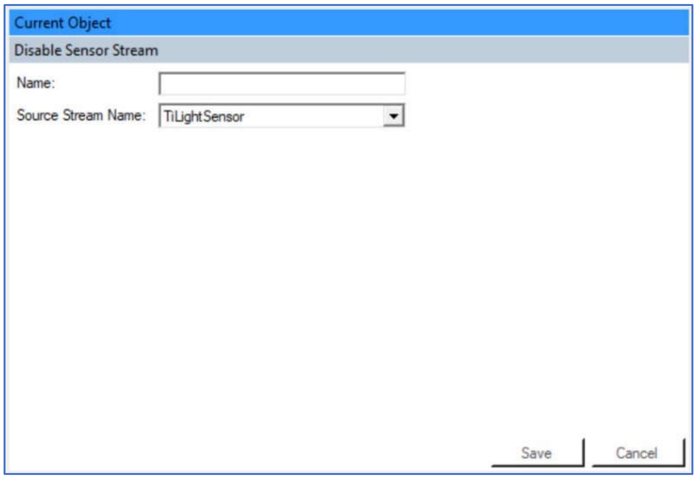
Figure 30: The Enable Sensor Stream Action object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Sensor Stream Name	This drop-down will present a list of all previously defined Sensor Stream objects.	Any drop-down item.

4.4.02 DisableSensorStream Action object

When executed, the DisableSensorStream Action object will disable a previously enabled Sensor Stream object. When disabled, the SoftHub application will attempt to disconnect from the specified Sensor Stream. Note that enabling Sensor Streams may be performed either via the *EnableSensorStream* Action object or by enabling "Auto Start All Sensors" in the Hub System Options screen.

The table below describes the available options and their valid value ranges.



Name:

Source Stream Name: TiLightSensor

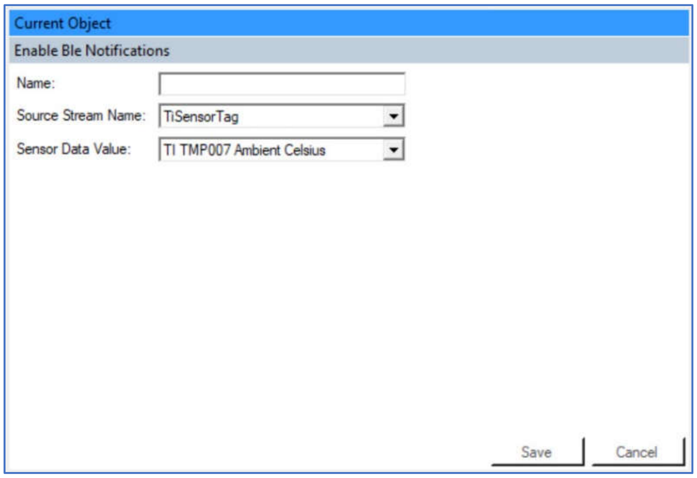
Figure 31: The Disable Sensor Stream Action object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Sensor Stream Name	This drop-down will present a list of all previously defined Sensor Stream objects.	Any drop-down item.

4.4.03 EnableBleNotifications Action object

When executed, the *EnableBleNotifications* object will enable notifications from the Bluetooth LE characteristic identified by the Source Stream and Sensor Data Value selected.

The table below describes the available options and their valid value ranges.



Name:

Source Stream Name: TiSensorTag

Sensor Data Value: TI TMP007 Ambient Celsius

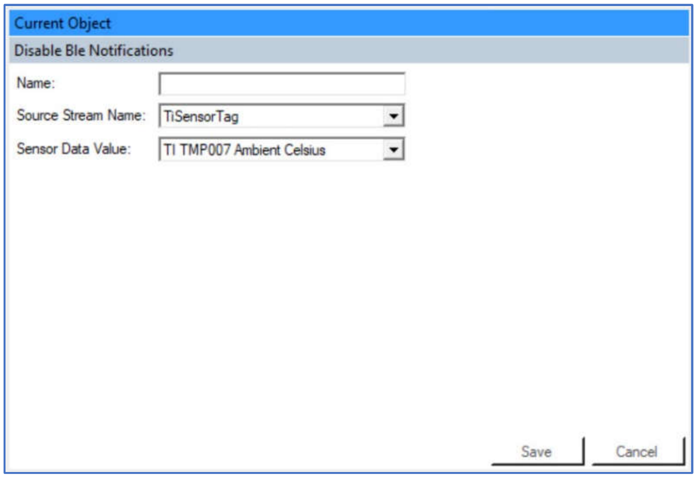
Figure 32: The Enable Ble Notifications Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Source Stream Name	This drop-down will present a list of all previously defined Ble Sensor Stream objects.	Any drop-down list entry.
Sensor Data Value	This drop-down list will contain each of the data value fields previously enabled for the selected Sensor Stream object which support "notify" operations.	Any drop-down list entry.

4.4.04 DisableBleNotifications Action object

When executed, the *DisableBleNotifications* object will disable further notifications from the Bluetooth LE characteristic identified by the Source Stream and Sensor Data Value selected.

The table below describes the available options and their valid value ranges.



Name:

Source Stream Name: TiSensorTag

Sensor Data Value: TI TMP007 Ambient Celsius

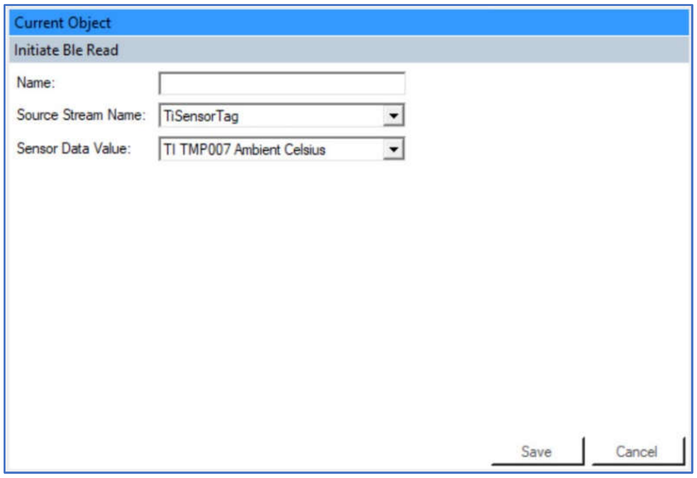
Figure 33: The Disable Ble Notifications Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Source Stream Name	This drop-down will present a list of all previously defined Ble Sensor Stream objects.	Any drop-down list entry.
Sensor Data Value	This drop-down list will contain each of the data value fields previously enabled for the selected Sensor Stream object which support "notify" operations.	Any drop-down list entry.

4.4.05 InitiateBleRead Action object

When executed, the *InitiateBleRead* object will initiate an asynchronous read operation for the Bluetooth LE characteristic identified by the Source Stream and Sensor Data Value selected.

The table below describes the available options and their valid value ranges.



Name:
Source Stream Name: TiSensorTag
Sensor Data Value: TI TMP007 Ambient Celsius

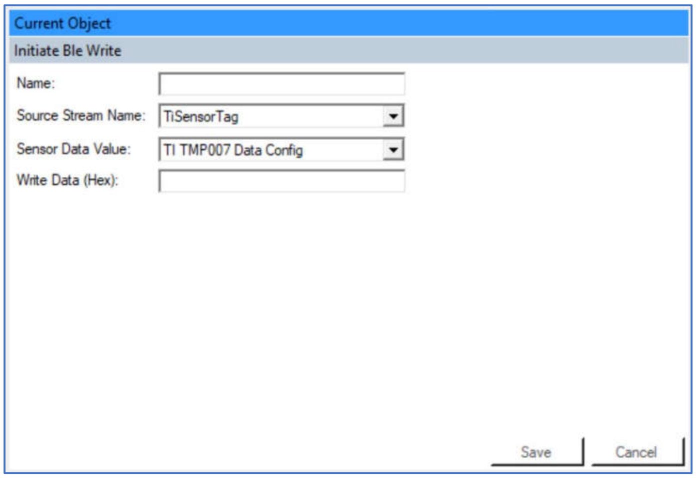
Figure 34: The Initiate Ble Read Action object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Source Stream Name	This drop-down will present a list of all previously defined Ble Sensor Stream objects.	Any drop-down list entry.
Sensor Data Value	This drop-down list will contain each of the data value fields previously enabled for the selected Sensor Stream object which support “read” operations.	Any drop-down list entry.

4.4.06 InitiateBleWrite Action object

hen executed, the *InitiateBleWrite* object will initiate an asynchronous write operation to the Bluetooth LE characteristic identified by the Source Stream and Sensor Data Value selected.

The table below describes the available options and their valid value ranges.



Name:
Source Stream Name: TiSensorTag
Sensor Data Value: TI TMP007 Data Config
Write Data (Hex):

Figure 345: The Initiate Ble Write Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Source Stream Name	This drop-down will present a list of all previously defined Ble Sensor Stream objects.	Any drop-down list entry.
Sensor Data Value	This drop-down list will contain each of the data value fields previously enabled for the selected Sensor Stream object which support "write" operations.	Any drop-down list entry.
Write Data (Hex)	In this textbox the user provides the hex representation of the data bytes to be written to the Ble characteristic (e.g. to write two bytes, "0x03" and "0x1F", to the characteristic you would enter 031F).	A valid string representing a series of hexadecimal bytes.

4.4.07 StartOneshotTimer Action object

When executed, the *StartOneshotTimer* Action object will start a "one shot" (i.e. non-repeating) timer. Timers are used typically to queue *TimerExpired* Event objects at expiration, which in turn may trigger the evaluation or one or more Rule objects. This timer is a "relative" timer, expiring at a set interval in the future. To set a timer for a specific absolute time in the future see the "Time of Day" timers. Note also that while timer values are specified in millisecond intervals some amount of drift is to be expected in non-realtime operating systems.

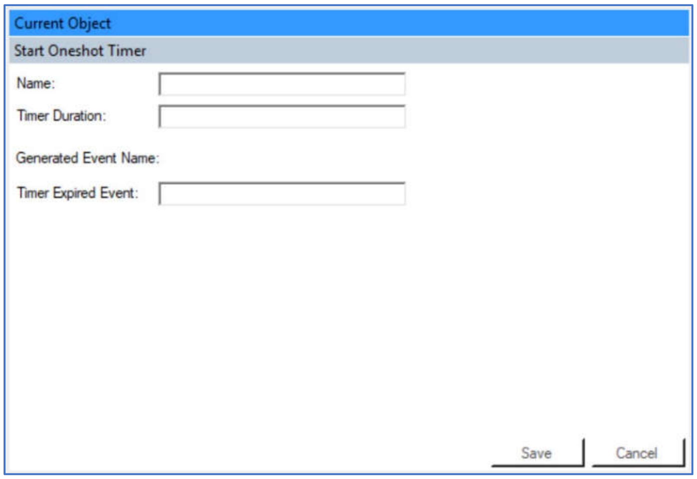


Figure 35: The Start Oneshot Timer Action object

The table below describes the available options and their valid value ranges.

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Timer Duration	The Timer Duration value indicates the number of milliseconds before the timer will expire.	Valid timer values are numbers between 50 and 18,446,744,073,709,551,615.
Timer Expired Event	This field names a *TimerExpired* Event object to be queued in the SoftHub's rule engine when the defined timer has expired.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.

4.4.08 StartContinuousTimer Action object

When executed, the *StartContinuousTimer* Action object will start a timer which will continuously restart itself once expired until a corresponding *StopTimer* Action object is executed in its name. Timers are used typically to queue *TimerExpired* Event objects at expiration, which in turn may trigger the evaluation or one or more Rule objects. This timer is a "relative" timer, expiring at a set interval in the future. To set a timer for a specific absolute time in the future see the "Time of Day" timers. Note also that while timer values are specified in millisecond intervals some amount of drift is to be expected in non-realtime operating systems.

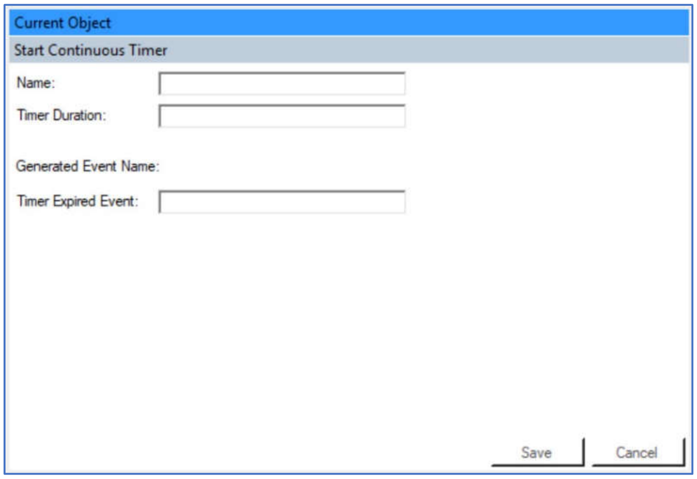


Figure 37: Start Continuous Timer Action object

The table below describes the available options and their valid value ranges.

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Timer Duration	The Timer Duration value indicates the number of milliseconds before the timer will expire.	Valid timer values are numbers between 50 and 18,446,744,073,709,551,615.
Timer Expired Event	This field names a *TimerExpired* Event object to be queued in the SoftHub's rule engine when the defined timer has expired.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.

4.4.09 StartOneShotTodTimer Action object

When executed, the *StartOneShotTODTimer* Action object will start a “one shot” (i.e. non-repeating) “time of day” timer. Timers are used typically to queue *TimerExpired* Event objects at expiration, which in turn may trigger the evaluation or one or more Rule objects. This timer is an “absolute” timer, expiring at a specific time in the future. Note also that time of day timers rely on the systems internal clock to be accurately set. This may not be possible on non-networked hub devices without access to NTP services.

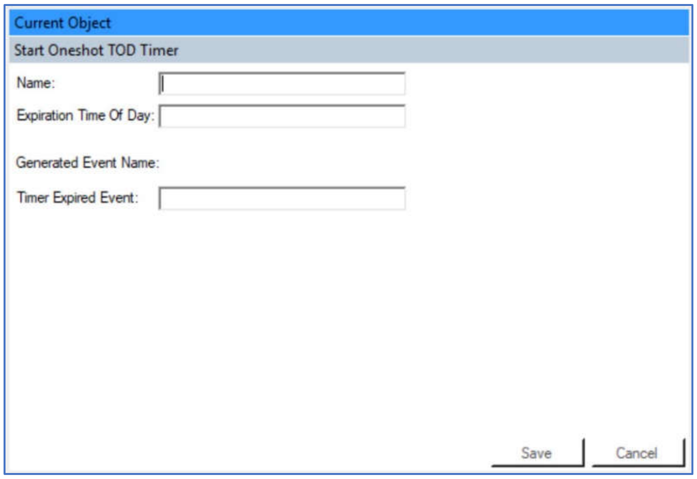


Figure 38: Start One Shot TOD Timer Action object

The table below describes the available options and their valid value ranges.

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Timer Time Of Day	This is the time of day value you wish the timer to expire.	Valid values must be specified in the 24 hour format of "HH:MM:SS" (e.g. "14:05:00" for 2:05 PM).
Timer Expired Event	This field names a *TimerExpired* Event object to be queued in the SoftHub's rule engine when the defined timer has expired.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.

4.4.10 StartContinuousTodTimer Action object

The *StartContinuousTodTimer* Action object will start a “time of day” timer which will expire at the same time every day until a corresponding *StopTimer* Action object is executed in its name. Timers are used typically to queue a *TimerExpired* Event objects at expiration, which in turn may trigger the evaluation or one or more Rule Objects. This timer is an “absolute” timer, expiring at a specific time in the future. Note also that time of day timers rely on the systems internal clock to be accurately set. This may not be possible on non-networked hub devices without access to NTP services.

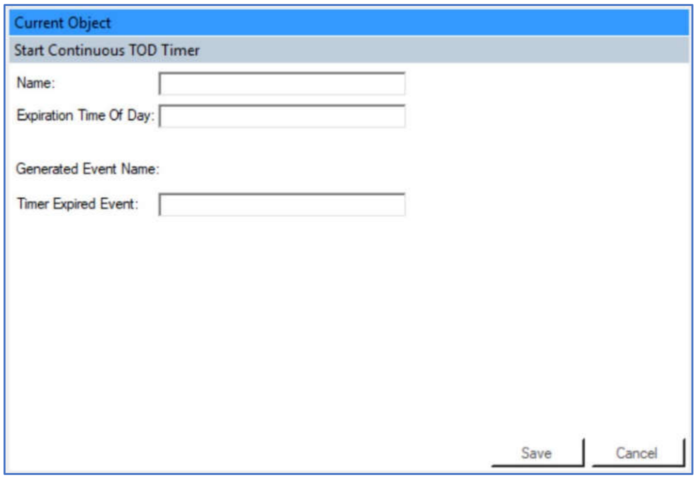


Figure 39: The Start Continuous TOD Timer Action object

The table below describes the available options and their valid value ranges.

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Timer Time Of Day	This is the time of day value you wish the timer to expire.	Valid values must be specified in the 24 hour format of "HH:MM:SS" (e.g. "14:05:00" for 2:05 PM).
Timer Expired Event	This field names a *TimerExpired* Event object to be queued in the SoftHub's rule engine when the defined timer has expired.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.

4.4.11 StopTimer Action object

Executing a *StopTimer* action will cancel a previously started timer and prohibit its associated *TimerExpired* Event object from being queued.

The table below describes the available options and their valid value ranges.

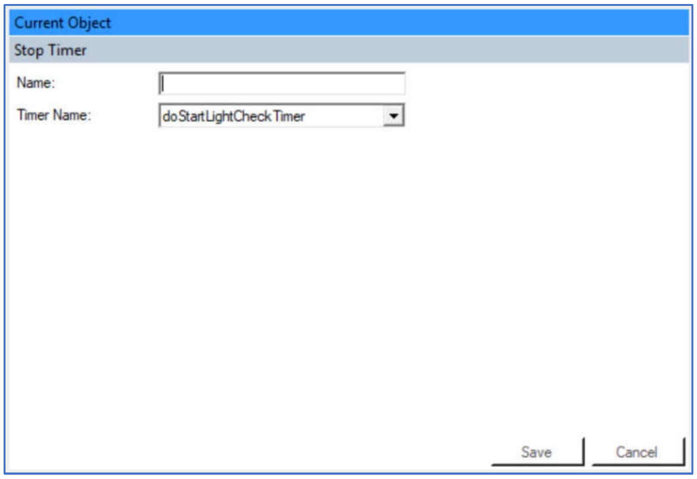

Current Object
Stop Timer
Name:
Timer Name: doStartLightCheckTimer

Figure 40: The Stop Timer Action object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Timer Name	This drop-down will present a list of all previously defined Timer objects. Note that Timer objects are automatically derived when a *StartOneShotTimer*, *StartContinuousTimer*, *StartOneShotTodTimer*, or *StartContinuousTodTimer* Action object is executed.	Any drop-down list entry.

4.4.12 SendData Action object

The *SendData* Action object will send the most recent raw data packet received from a Sensor Stream to a remote packet capture server over a TLS encrypted connection. Typically the remote server will be running the OmnIoT Remote Packet Capture Service which will receive and decode the incoming packets. To ensure zero packet loss the user should enable packet caching.

The table below describes the available options and their valid value ranges.

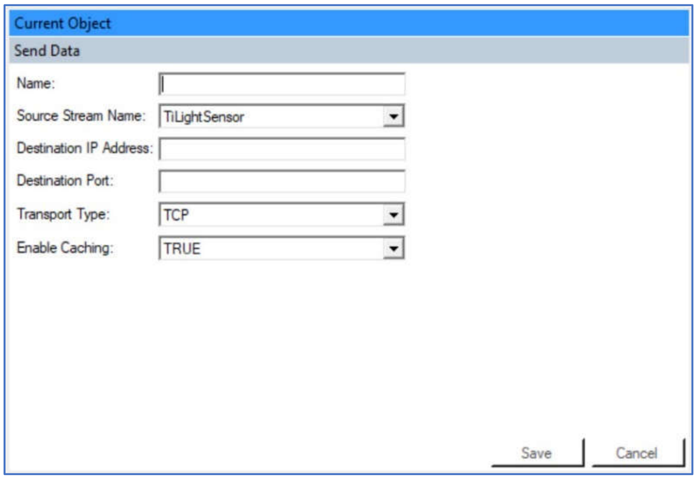

Current Object
Send Data
Name:
Source Stream Name: TiLightSensor
Destination IP Address:
Destination Port:
Transport Type: TCP
Enable Caching: TRUE

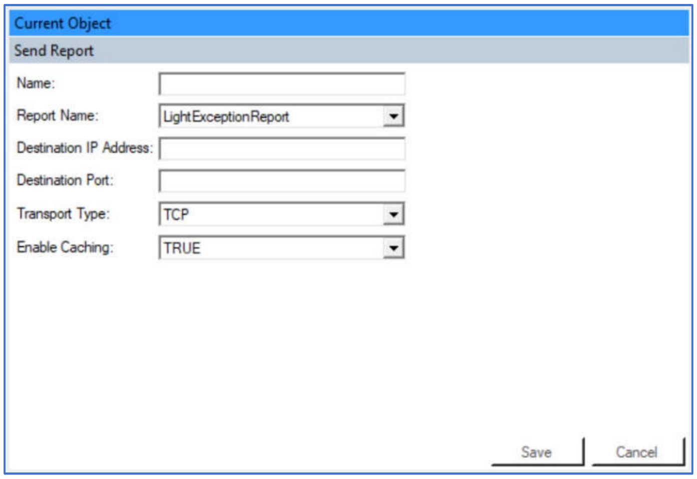
Figure 41: The Send Data Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Source Stream Name	This drop-down will present a list of all previously defined Sensor Stream objects.	Any drop-down list entry.
Destination IP	This field is the IP address of the remote packet capture server.	Any valid Ipv4 or Ipv6 format IP address.
Destination Port	This field indicates the port number on the remote server which the packet receiver will be listening on.	A valid integer in the range of 1 to 65,535.

4.4.13 SendReport Action object

When executed, the *SendReport* Action object will trigger the assembling of a previously defined *UserReport* Report object (custom packet) which in turn will be forwarded to the specified remote server over a TLS encrypted connection. Typically the remote server will be running the OmnIoT Remote Packet Capture Service which will receive and decode the incoming packets. To ensure zero packet loss the user should enable packet caching.

The table below describes the available options and their valid value ranges.



Name:

Report Name: LightExceptionReport

Destination IP Address:

Destination Port:

Transport Type: TCP

Enable Caching: TRUE

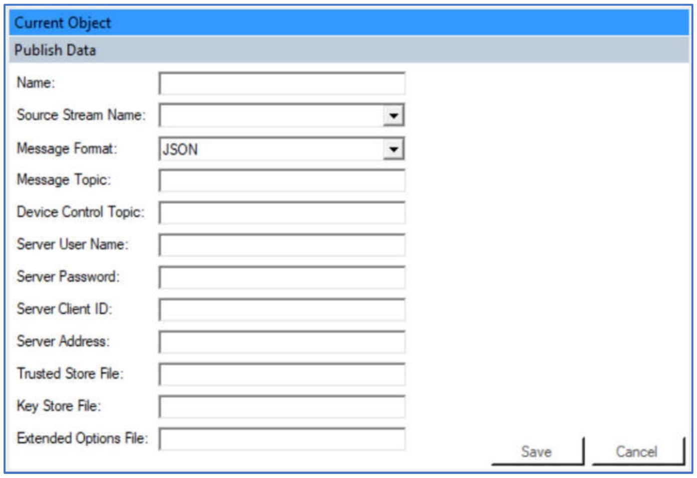
Figure 42: The Send Report Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Source Report Name	This drop-down will present a list of all previously defined UserReport Report objects.	Any drop-down list entry.
Destination IP	This field is the IP address of the remote packet capture server.	Any valid Ipv4 or Ipv6 format IP address
Destination Port	This field indicates the port number on the remote server which the packet receiver will be listening on.	A valid integer in the range of 1 to 65,535.

4.4.14 PublishData Action object

The *PublishData* Action object will publish the most recent raw data packet received from a Sensor Stream to a remote MQTT broker. The SoftHub is compatible with most third-party cloud based platforms providing an MQTT interface. To ensure zero packet loss the user should enable packet caching.

The table below describes the available options and their valid value ranges.



Name:
Source Stream Name:
Message Format: JSON
Message Topic:
Device Control Topic:
Server User Name:
Server Password:
Server Client ID:
Server Address:
Trusted Store File:
Key Store File:
Extended Options File:

Figure 43: The Publish Data Action object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Source Stream Name	This drop-down will present a list of all previously defined Sensor Stream objects.	Any drop-down list entry.
Message Format	Dictates the format the message will be published in.	Any drop-down list entry.
Message Topic	Specifies the topic the message will be published to.	Valid values for this field include any printable characters up to 256 bytes in length.
Device Control Topic	Specifies the topic for the SoftHub to monitor for incoming remote control packets.	Valid values for this field include any printable characters up to 256 bytes in length.
Server User Name	The login name to be used when connecting to the remote MQTT broker	Valid values for this field include any printable characters up to 256 bytes in length.
Server Password	The login password to be used when connecting to the remote MQTT broker	Valid values for this field include any printable characters up to 256 bytes in length.
Server Client ID	The Client ID to be used when connecting to the remote MQTT broker	Valid values for this field include any printable characters up to 256 bytes in length.
Server Address	The server address of the target remote MQTT broker.	Valid values for this field include any printable characters up to 256 bytes in length.
Trusted Store File	The local SSL trusted store file name to be used.	Valid values for this field include any printable characters up to 256 bytes in length.
Key Store File	The local SSL key store file name to be used.	Valid values for this field include any printable characters up to 256 bytes in length.
Extended Options File	The name of an Extended Options File for providing additional connection parameters.	Valid values for this field include any alpha-numeric characters up to 64 bytes in length. Note that when specifying an Extended Options File, the filetype (.mcfg) should be omitted.

4.4.15 PublishReport Action object

When executed, the *PublishReport* Action object will trigger the assembling of a previously defined *UserReport* Report object (custom packet) which in turn will be published to the specified remote MQTT broker. The SoftHub is compatible with most third-party cloud based platforms providing an MQTT interface. To ensure zero packet loss the user should enable packet caching.

The table below describes the available options and their valid value ranges.

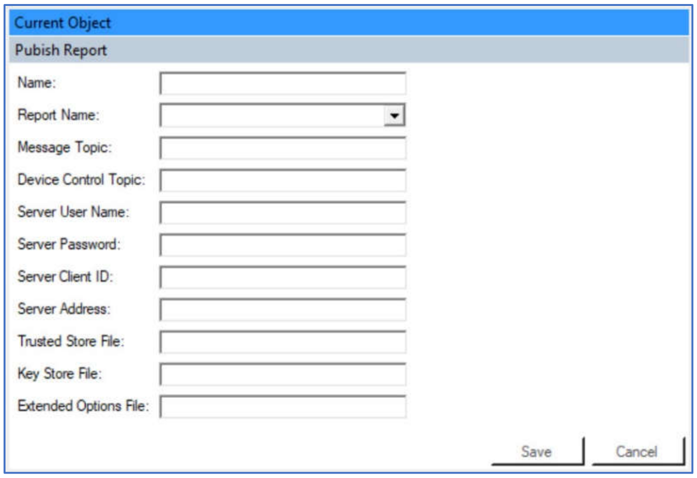


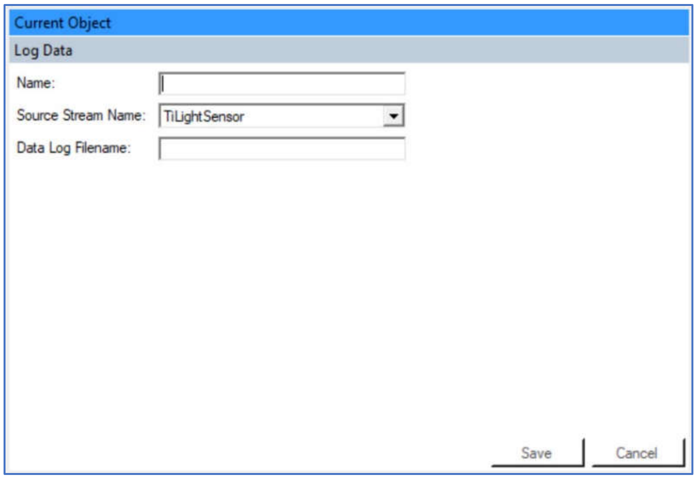
Figure 44: The Publish Report Action object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Source Report Name	This drop-down will present a list of all previously defined UserReport Report objects.	Any drop-down list entry.
Message Topic	Specifies the topic the message will be published to.	Valid values for this field include any printable characters up to 256 bytes in length.
Device Control Topic	Specifies the topic for the SoftHub to monitor for incoming remote control packets.	Valid values for this field include any printable characters up to 256 bytes in length.
Server User Name	The login name to be used when connecting to the remote MQTT broker	Valid values for this field include any printable characters up to 256 bytes in length.
Server Password	The login password to be used when connecting to the remote MQTT broker	Valid values for this field include any printable characters up to 256 bytes in length.
Server Client ID	The Client ID to be used when connecting to the remote MQTT broker	Valid values for this field include any printable characters up to 256 bytes in length.
Server Address	The server address of the target remote MQTT broker.	Valid values for this field include any printable characters up to 256 bytes in length.
Trusted Store File	The local SSL trusted store file name to be used.	Valid values for this field include any printable characters up to 256 bytes in length.
Key Store File	The local SSL key store file name to be used.	Valid values for this field include any printable characters up to 256 bytes in length.
Extended Options File	The name of an Extended Options File for providing additional connection parameters.	Valid values for this field include any alpha-numeric characters up to 64 bytes in length. Note that when specifying an Extended Options File, the filetype (.mcfg) should be omitted.

4.4.16 LogData Action object

When executed, the *LogData* Action object will write the raw packet most recently received from the referenced Sensor Stream to a logfile on the hub devices' local storage. If desired, the default log file directory can be overridden in the System Options screen.

The table below describes the available options and their valid value ranges.



Name:
Source Stream Name: TiLightSensor
Data Log Filename:

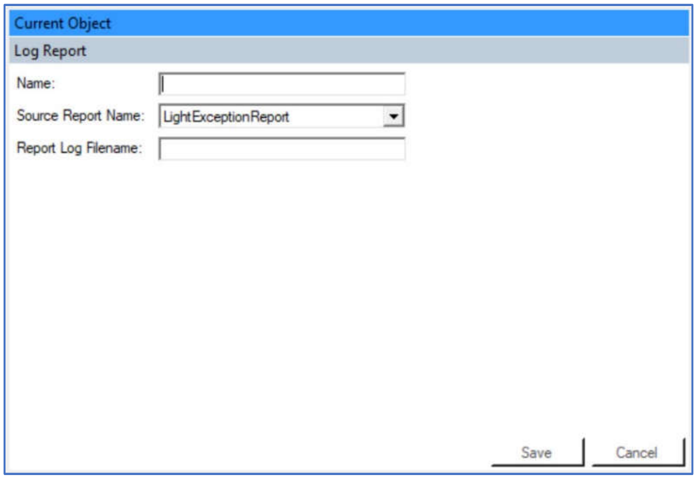
Figure 45: The Log Data Action object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Source Stream Name	This drop-down will present a list of all previously defined Sensor Stream objects.	Any drop-down list entry.
Data Log Filename	The data log file name specifies the log file that the data will be written to in the log file directory. Note that the name chosen will be automatically suffixed with ".log".	Must be an alphanumeric string between 2 and 40 characters in length.

4.4.17 LogReport Action object

When executed, the *LogReport* Action object will trigger the assembling of a previously defined *UserReport* Report object (custom packet) which in turn will be logged to a logfile on the hub devices' local storage. If desired, the default log file directory can be overridden in the System Options screen.

The table below describes the available options and their valid value ranges.



Name:
Source Report Name: LightExceptionReport
Report Log Filename:

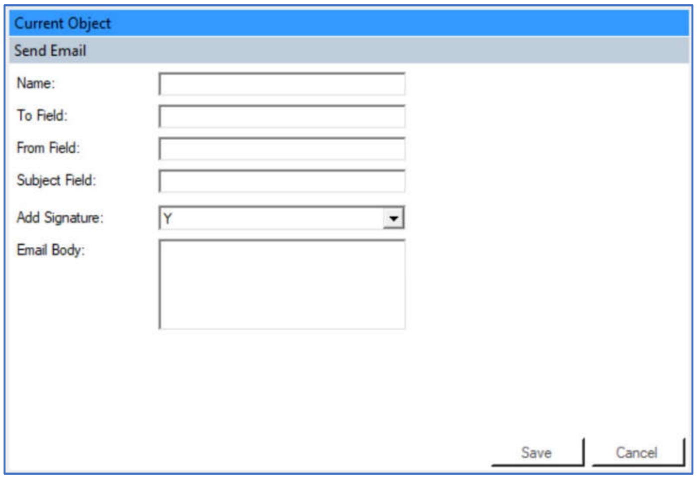
Figure 46: The Log Report Action object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Source Report Name	This drop-down will present a list of all previously defined UserReport Report objects.	Any drop-down list entry.
LogReport Filename	The data log file name specifies the log file that the data will be written to in the log file directory. Note that the name chosen will be automatically suffixed with ".log".	Must be an alphanumeric string between 2 and 40 characters in length.

4.4.18 SendEmail Action object

When executed, the SendEmail Action object will queue an email or SMS message to be sent to listed recipient(s). Note that these objects rely on the SMTP Email parameters in the SoftHub's Email System Options screen having been configured correctly.

The table below describes the available options and their valid value ranges.



Add Signature: Y

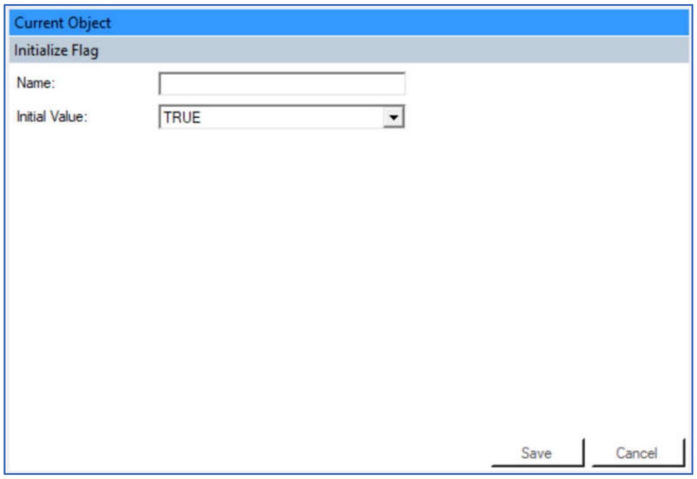
Figure 47: The Send Email Action object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
To Field	This field must contain one or more email addresses in a valid email format. If more than one address is specified each address should be separated by a semicolon.	Limited to the character set of all alpha-numerics, plus any of the following "=-.,:/@!#$%&*+^_{}~".
From Field	This field should reflect the email address related to the SMTP login credentials specified in the Email System Options configuration section.	Character set limited as above.
Subject Field	This field will be sent as the subject.	Character set limited as above, length limited to 0 to 40 characters.
Add Signature	This field determines whether the system will append a signature to the message indicating the time it was produced as well as the originating hub ID.	Any drop-down list entry.
Email Body	This field will be sent as the email (or SMS) body.	Character set limited as above, length limited to 0 to 140 characters.

4.4.19 Initialize Flag Action object

When executed, the *InitializeFlag* Action object will create and initialize a new Flag object. Once Flag objects have been created they can be set, cleared, or toggled with the *SetFlag*, *ClearFlag*, and *ToggleFlag* Action objects. Flag object states are also typically incorporated into Rule object conditional statements via the *FlagState* State object.

The table below describes the available options and their valid value ranges.



Name:

Initial Value: TRUE

Figure 48: The Initialize Flag Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Initial Value	This drop-down allows the user to set the initial value of the flag being created to either TRUE or FALSE.	Any drop-down list entry.

4.4.20 SetFlag Action object

When executed, the *SetFlag* Action object will set a previously created Flag object to a “true” state. Flag object states may be incorporated into Rule object conditional statements via the *FlagState* State object.

The table below describes the available options and their valid value ranges.

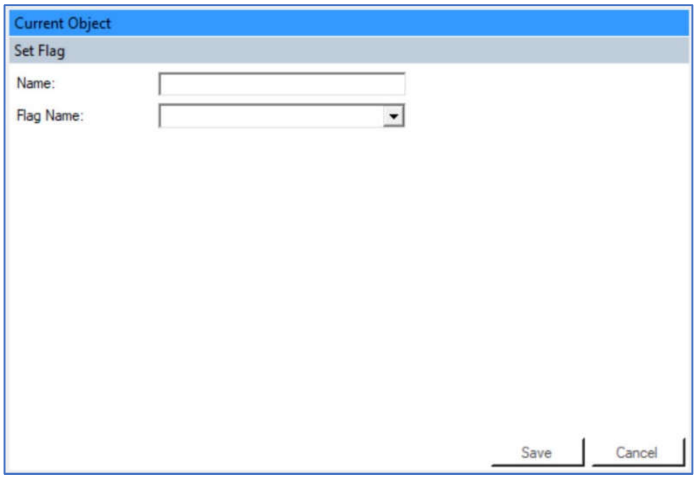


Figure 49: The Set Flag Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Flag Name	This drop-down will present a list of all previously defined Flag objects. Note that Flag objects are automatically derived when a new *InitializeFlag* Action object is executed.	Any drop-down list entry.

4.4.21 ClearFlag Action object

When executed, the *ClearFlag* Action object will set a previously created Flag object to a "false" state. Flag object states may be incorporated into Rule object conditional statements via the *FlagState* State object.

The table below describes the available options and their valid value ranges.

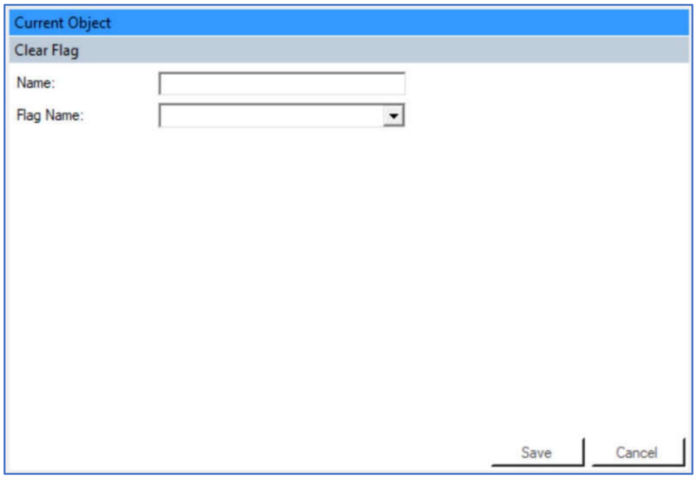


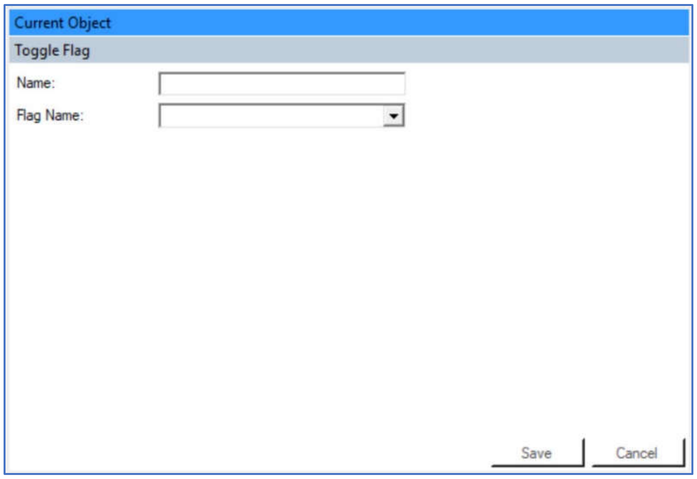
Figure 50: The Clear Flag Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Flag Name	This drop-down will present a list of all previously defined Flag objects. Note that Flag objects are automatically derived when a new *InitializeFlag* action object is executed.	Any drop-down list entry.

4.4.22 ToggleFlag Action object

When executed, the *ToggleFlag* Action object will invert a previously created Flag objects' current logical state. Flag object states may be incorporated into Rule object conditional statements via the *FlagState* State object.

The table below describes the available options and their valid value ranges.



Current Object

Toggle Flag

Name:

Flag Name:

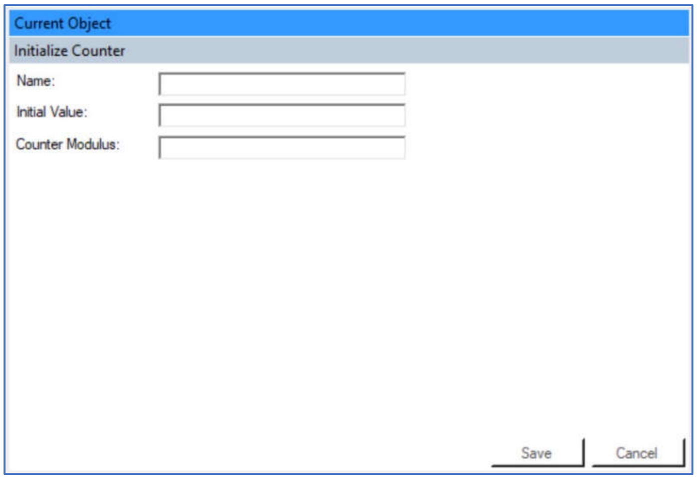
Figure 51: The Toggle Flag Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Flag Name	This drop-down will present a list of all previously defined Flag objects. Note that Flag objects are automatically derived when a new *InitializeFlag* action object is executed.	Any drop-down list entry.

4.4.23 InitializeCounter Action object

When executed, the *InitializeCounter* Action object will create and initialize a new Counter object. Once a Counter object has been created it may be incremented or decremented with the *IncrementCounter* and *DecrementCounter* Action objects. Counter object states may also be incorporated into Rule object conditional statements via the *CounterValue* State object.

The table below describes the available options and their valid value ranges.



Current Object

Initialize Counter

Name:

Initial Value:

Counter Modulus:

Figure 52: The Initialize Counter Action object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Initial Value	The Initial Value field defines the value that the created counter will be set to initially.	Valid values include signed integer numbers between –2,147,483,648 and 2,147,483,647. Note that the Initial Value field's absolute value must be less than any specified Counter Modulus value (unless the modulus is 0).
Counter Modulus	The Counter Modulus field allows you to set a modulus value for the counter.	Valid values include unsigned integers between 0 and 2,147,483,647. To disable modulus processing you can set this value to 0.

4.4.24 IncrementCounter Action object

When executed, the *IncrementCounter* Action object will increment a previously created Counter object by the specified value. Counter object states may be incorporated into Rule object conditional statements via the *CounterValue* State object.

The table below describes the available options and their valid value ranges.

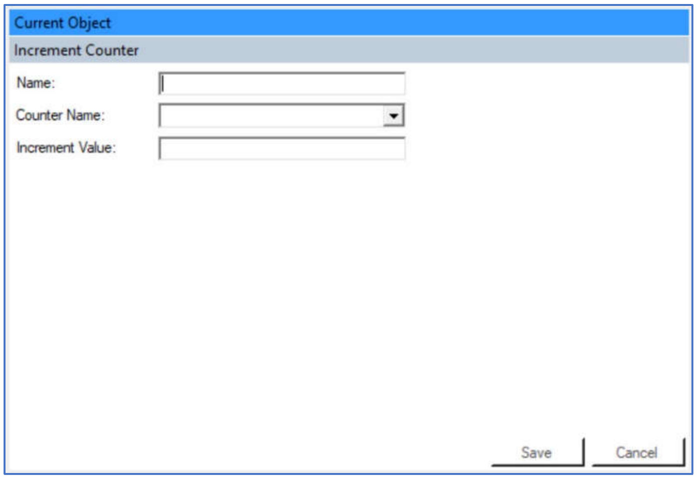


Figure 53: The Increment Counter Action object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Counter Name	This drop-down will present a list of all previously defined Counter objects. Note that Counter objects are automatically derived any time a new *InitializeCounter* Action object is executed.	Any drop-down list entry.
Increment Value	This value represents the value that will be added to the counter when the IncrementCounter Action object is executed.	Valid values include signed integer numbers between –2,147,483,648 and 2,147,483,647.

4.4.25 DecrementCounter Action object

When executed, the *DecrementCounter* Action object will decrement a previously created Counter object by the specified value. Counter object states may be incorporated into Rule object conditional statements via the *CounterValue* State object.

The table below describes the available options and their valid value ranges.

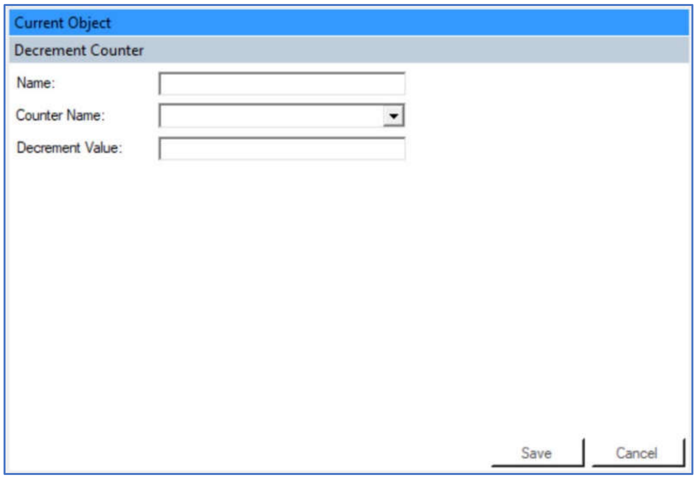
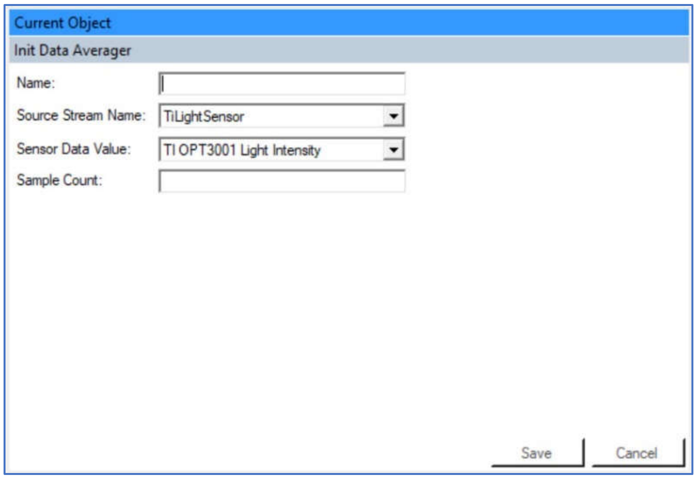


Figure 54: The Decrement Counter Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Counter Name	This drop-down will present a list of all previously defined Counter objects. Note that Counter objects are automatically derived any time a new *InitializeCounter* action object is executed.	Any drop-down list entry.
Decrement Value	This value represents the value that will be subtracted from the counter when the DecrementCounter Action object is executed.	Valid values include signed integer numbers between –2,147,483,648 and 2,147,483,647.

4.4.26 InitDataAverager Action object

When executed, the *InitDataAverager* Action object will create and initialize a new object capable of maintaining an average of an incoming sensor value. Data averager objects can be used in conjunction with the *UpdateDataAverager* Action object to maintain either a running or a moving average of a selected sensor data value. In turn, the value of the data averager object may be evaluated as part of a Rule object's conditional logic via the *SensorDataAveragerValue* State object. Data averager values may also be included as fields in custom *UserReport* Report objects.



Name:

Source Stream Name: TiLightSensor

Sensor Data Value: TI OPT3001 Light Intensity

Sample Count:

Figure 55: The Init Data Averager Action object

The table below describes the available options and their valid value ranges.

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Source Stream Name	This drop-down will present a list of all previously defined Sensor Stream objects.	Any drop-down list entry.
Sensor Data Value	This drop-down list will contain each of the data value fields previously enabled for the selected Sensor Stream object.	Any drop-down list entry.
Sample Count	This value represents the number of samples to use as a maximum when calculating the average. Note that prior to reaching the full sample count an average of the "n" samples currently available will be provided.	For moving averages this is a positive integer value between 2 and 2500. For a running average this value should be 0.

4.4.27 UpdateDataAverager Action object

When executed, the *UpdateDataAverager* object will cause a new value to be averaged into in an averager object previously created via an *InitDataAverager* Action object. The new value will be the latest available Sensor Stream data value originally referenced in the InitDataAverager object definition.

The table below describes the available options and their valid value ranges.

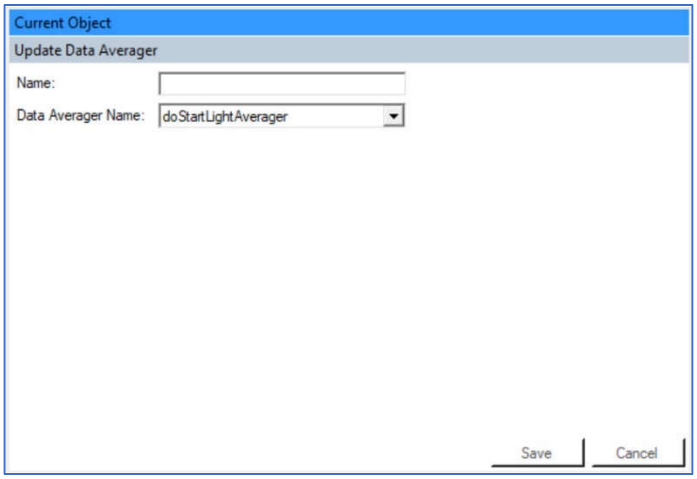

Current Object
Update Data Averager
Name:
Data Averager Name: doStartLightAverager

Figure 56: The Update Data Averager Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Data Averager Name	This drop-down will present a list of all previously defined Sensor Data Averager objects. Note that Sensor Data Averager objects are automatically derived any time a new "Initialize Data Averager" action object is executed.	Any drop-down list entry.

4.4.28 InitSensorDataCache Action object

When executed, the *InitSensorDataCache* Action object will create and initialize a new sensor data cache object. These cache objects can be used in conjunction with the *UpdateSensorDataCache* Action object to maintain a running cache of a selected sensor data value. In turn, Rule object conditional logic can reference the *NumberOfCachedSamples* State object to conditionally execute its list of Action objects. Sensor data cache values may also be included as fields in custom *UserReport* Report objects. Note also that this action object may be re-executed at any time to reset and clear the created cache object.

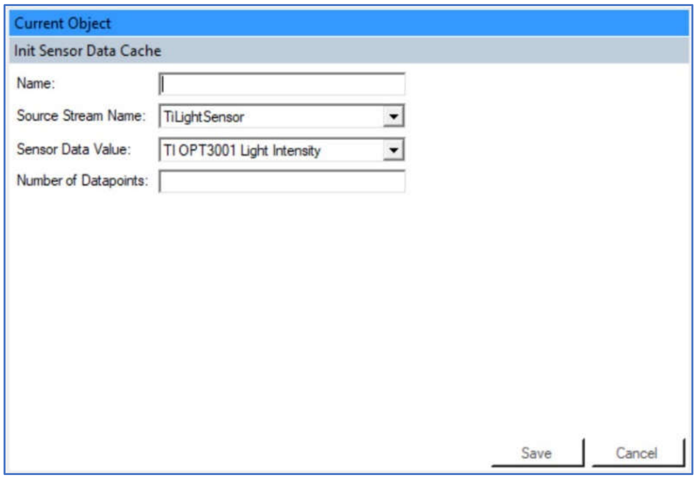

Current Object
Init Sensor Data Cache
Name:
Source Stream Name: TiLightSensor
Sensor Data Value: TI OPT3001 Light Intensity
Number of Datapoints:

Figure 57: The Init Sensor Data Cache Action object

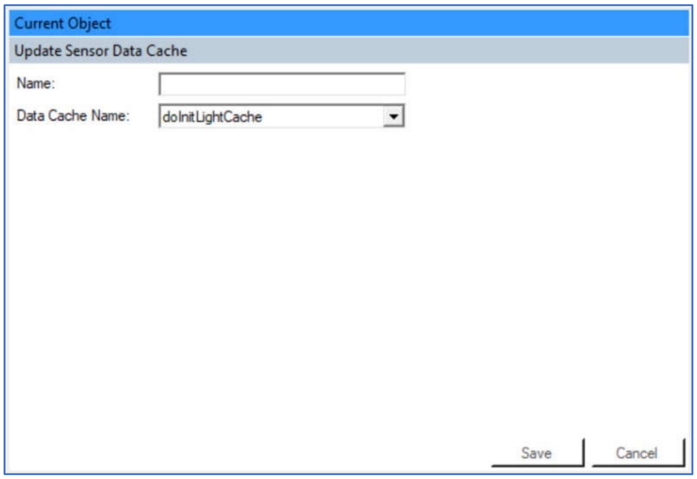
The table below describes the available options and their valid value ranges.

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Source Stream Name	This drop-down will present a list of all previously defined Sensor Stream objects.	Any drop-down list entry.
Sensor Data Value	This drop-down list will contain each of the data value fields previously enabled for the selected Sensor Stream object.	Any drop-down list entry.
Cache Size	This number represents the maximum number of values the cash will hold it in given time. Note that data cache objects operate as a circular cache with new cached values overwriting the oldest cached values once the cache is full.	A numeric value between 2 and 1024.

4.4.29 UpdateSensorDataCache Action object

The *UpdateSensorDataCache* object will take the latest received sensor data value identified by the associated *InitSensorDataCache* object and add it to the cache buffer.

The table below describes the available options and their valid value ranges.



Data Cache Name: doInitLightCache

Figure 58: The Update Sensor Data Cache Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Data Cache Name	This drop-down will present a list of all previously defined Sensor Data Cache objects. Note that Sensor Data Cache objects are automatically derived any time a new *InitSensorDataCache* action object is created.	Any drop-down list entry.

4.4.30 SetGpioPinLow Action object

When executed, the *SetGpioPinLow* Action object will set a GPIO pin that has been enabled as an output pin in the SoftHub's System Options to "Low".

The table below describes the available options and their valid value ranges.

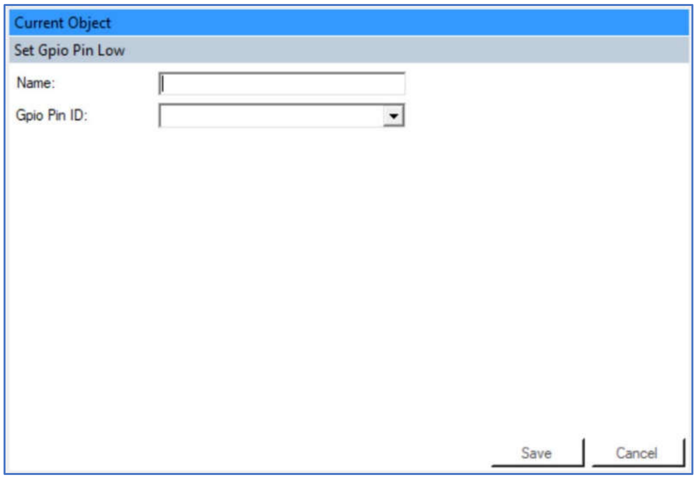


Figure 36: The Set GPIO Pin Low Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
GPIO Pin Id	This drop-down lists the GPIO Output Pins that have been previously enabled in the SoftHub's GPIOs System Options configuration screen.	Any drop-down list entry.

4.4.31 SetGpioPinHigh Action object

When executed, the *SetGpioPinHigh* Action object will set a GPIO pin that has been enabled as an output pin in the SoftHub's System Options to "High".

The table below describes the available options and their valid value ranges.

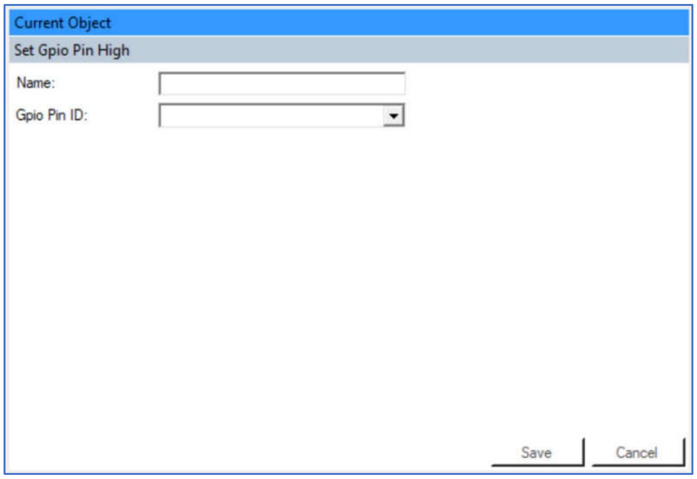


Figure 60: The Set GPIO Pin High Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
GPIO Pin Id	This drop-down lists the GPIO Output Pins that have been previously enabled in the SoftHub's GPIOs System Options configuration screen.	Any drop-down list entry.

4.4.32 ToggleGpioPin Action object

When executed, the *ToggleGpioPin* Action object will set a GPIO pin that has been enabled as an output pin in the SoftHub’s System Options to its inverted state.

The table below describes the available options and their valid value ranges.

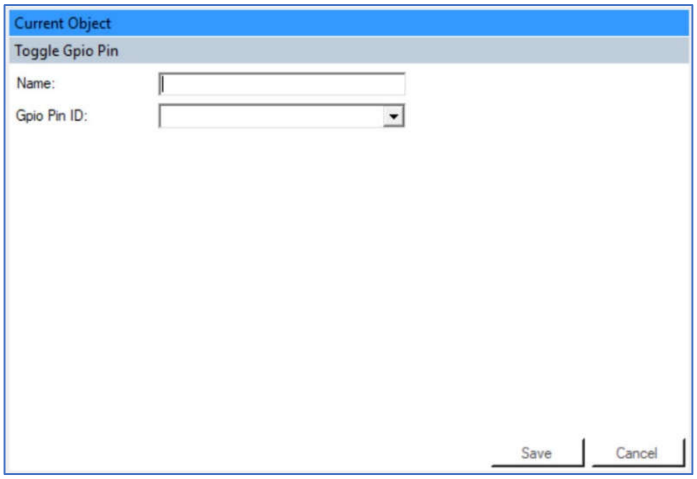


Figure 61: The Toggle GPIO Pin Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
GPIO Pin Id	This drop-down lists the GPIO Output Pins that have been previously enabled in the SoftHub's GPIOs System Options configuration screen.	Any drop-down list entry.

4.4.33 ExecSysCommand Action object

When executed, the *ExecSysCommand* Action object will initiate the execution of a command on the system in the background. Multiple commands may be initiated at once and may be specified to run concurrently or consecutively. *These objects should be used with great care* as they will be executed with the system's highest privilege/authority level (i.e. "root" or "Admin" depending on the platform). You should carefully review the parameters described below before incorporating these objects in your configuration files.

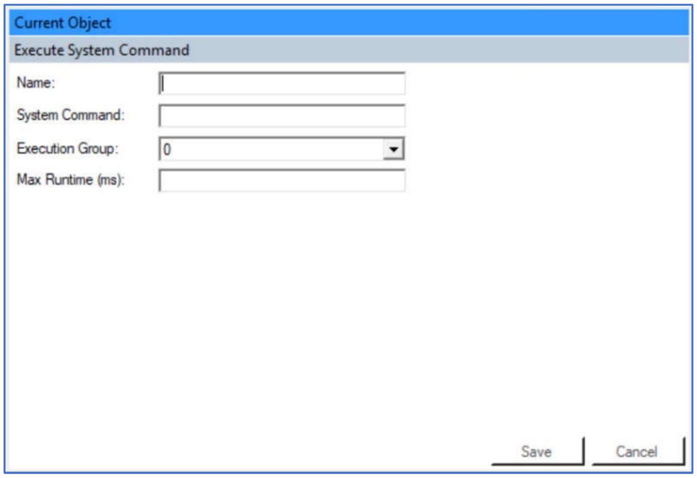

Current Object
Execute System Command
Name:
System Command:
Execution Group: 0
Max Runtime (ms):

Figure 62: The Execute System Command Action object

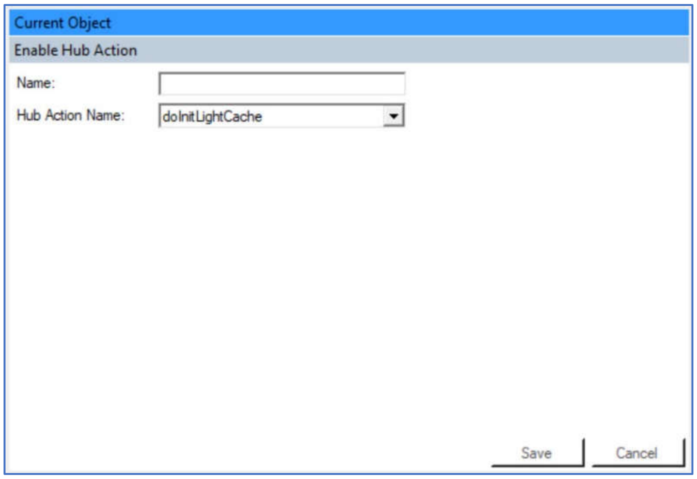
The table below describes the available options and their valid value ranges.

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
System Command	This is the command to be executed. You can execute any command as you would at a system line command prompt. Special formatting may be required based on the target platform.	On Linux systems the command, and each of its parameters, must be enclosed in a separate pair of double quotes (see the execv man page for details). An example of a copy command would be all the following characters between (but not including) the square brackets ["/bin/cp" "/home/test/myfile" "/home/test/myfile.sav"].
Execution Group	The execution group allows commands to be run either concurrently or consecutively. Commands in group 0 will always be executed immediately regardless of whether earlier executed commands have completed or not. Commands in groups 1 through 5 will be queued into individual queues and will be executed consecutively within their queue on a first in first out basis. For instance, if you wanted to play a series of sound files you could queue them into queue 1 to ensure that only one would ever be played at the time.	Any drop-down list entry.
Max Runtime (ms)	This is the maximum runtime in milliseconds a command can be run before being killed. Once the runtime has been exceeded a soft-kill (sigterm) will be issued, if the command has not terminated in the following 5 seconds a hard-kill (sigkill) will be sent. Note that when the SoftHub shuts down it will attempt to terminate any outstanding commands still running as well.	Any positive number between 0 and 9,223,372,036,854,775,807. A max runtime of 0 essentially sets no time limit on execution.

4.4.34 EnableHubAction Action object

When executed, the *EnableHubAction* Action object will enable a previously defined Action object. When disabled, an Action object will no longer be executed regardless of whether a refering Rule object's conditional logic is satisfied or not. Note that all action objects are enabled by default until explicitly disabled by either a *DisableHubAction* or *ToggleHubAction* Action object.

The table below describes the available options and their valid value ranges.



Name:

Hub Action Name: doInitLightCache

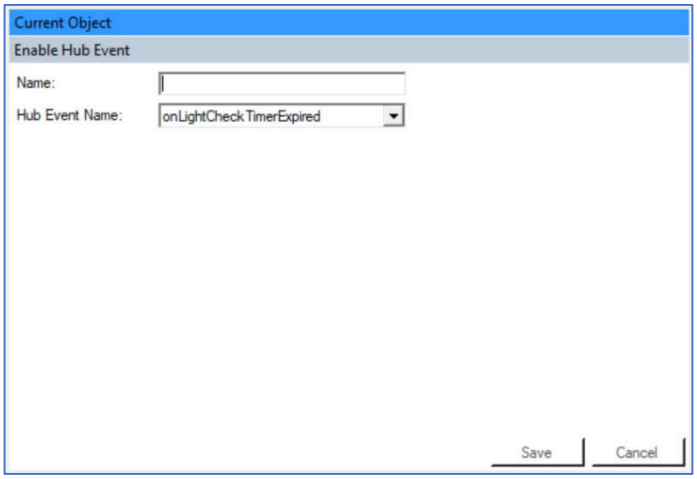
Figure 63: The Enable Hub Action Action object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Hub Action Name	This drop-down will present a list of all previously defined Action objects excluding the *EnableHubAction*, *DisableHubAction*, and *ToggleHubAction* Action objects themselves.	Any drop-down list entry.

4.4.35 EnableHubEvent Action object

When executed, the *EnableHubEvent* Action object will enable a previously defined Event object. When disabled, Event objects will still be queued into the rule engine's event queue but will be immediately discarded when received (an no associated Rule objects will be evaluated). Note that all event objects are enabled by default until explicitly disabled by either a *DisableHubEvent* or *ToggleHubEvent* Action object.

The table below describes the available options and their valid value ranges.



Name:

Hub Event Name: onLightCheckTimerExpired

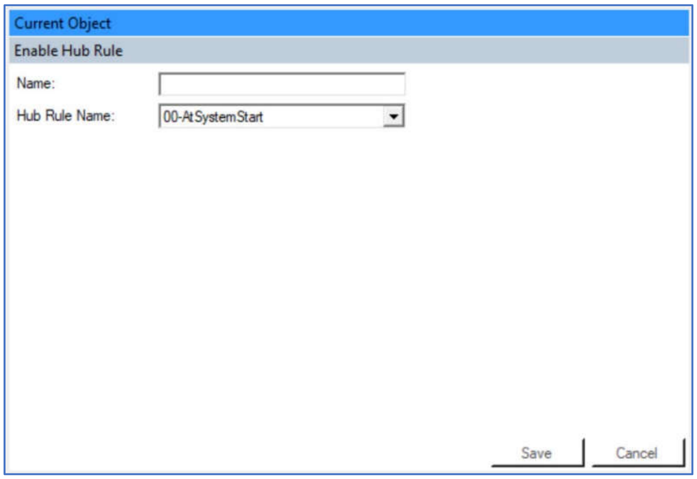
Figure 64: The Enable Hub Event Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Hub Event Name	This drop-down will present a list of all previously defined Event objects.	Any drop-down list entry.

4.4.36 EnableHubRule Action object

When executed, the *EnableHubRule* Action object will enable a previously defined Rule object. When disabled, a Rule object will no longer be evaluated/executed when any associated Event object is received by the rule engine. Note that all rule objects are enabled by default until explicitly disabled by either a *DisableHubRule* or *ToggleHubRule* Action object.

The table below describes the available options and their valid value ranges.



Name:

Hub Rule Name: 00-AtSystemStart

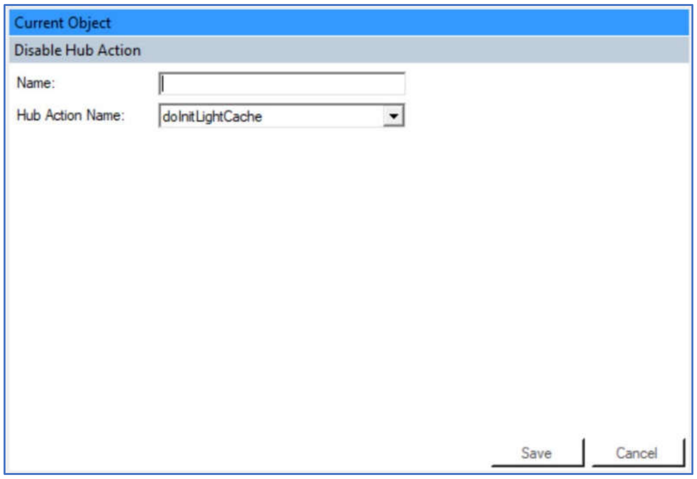
Figure 65: The Enable Hub Rule Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Hub Rule Name	This drop-down will present a list of all previously defined Rule objects.	Any drop-down list entry.

4.4.37 DisableHubAction Action object

When executed, the *DisableHubAction* Action object disable a previously defined Action object. When disabled, an Action object will no longer be executed regardless of whether a refering Rule object's conditional logic is satisfied or not. Note that all action objects are enabled by default until explicitly disabled by either a *DisableHubAction* or *ToggleHubAction* Action object.

The table below describes the available options and their valid value ranges.



Name:

Hub Action Name: doInitLightCache

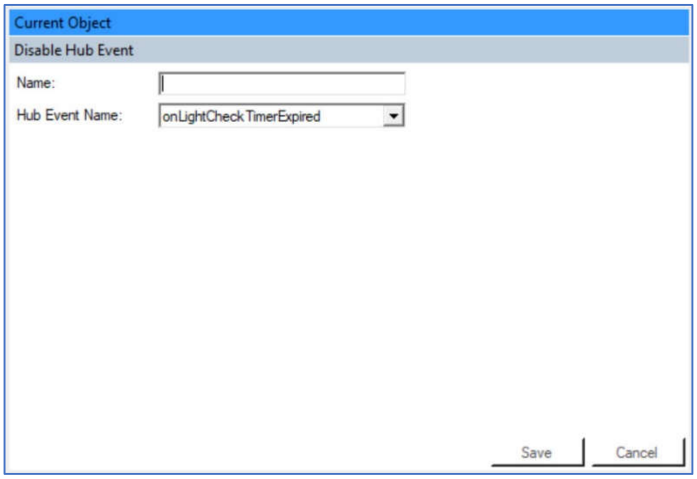
Figure 66: The Disable Hub Action Action object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Hub Action Name	This drop-down will present a list of all previously defined Action objects excluding the *EnableHubAction*, *DisableHubAction*, and *ToggleHubAction* Action objects themselves.	Any drop-down list entry.

4.4.38 DisableHubEvent Action object

When executed, the *DisableHubEvent* Action object will disable a previously defined Event object. When disabled, Event objects will still be queued into the rule engine's event queue but will be immediately discarded when received (an no associated Rule objects will be evaluated). Note that all event objects are enabled by default until explicitly disabled by either a *DisableHubEvent* or *ToggleHubEvent* Action object.

The table below describes the available options and their valid value ranges.



Name:

Hub Event Name: onLightCheckTimerExpired

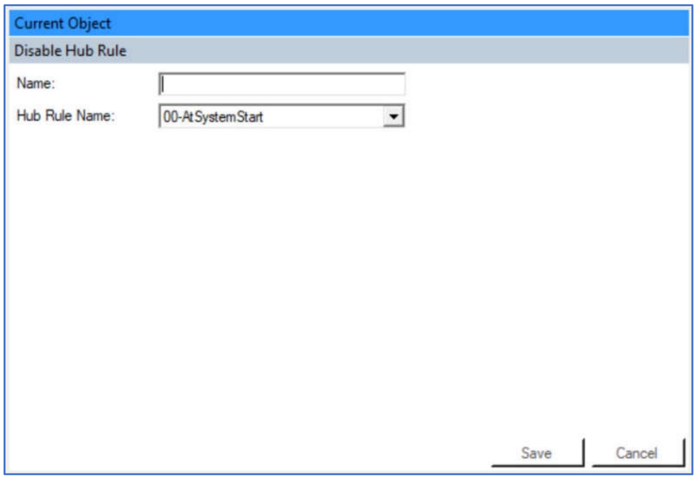
Figure 67: The Disable Hub Event Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Hub Event Name	This drop-down will present a list of all previously defined Event objects.	Any drop-down list entry.

4.4.39 DisableHubRule Action object

When executed, the *DisableHubRule* Action object will disable a previously defined Rule object. When disabled, a Rule object will no longer be evaluated/executed when any associated Event object is received by the rule engine. Note that all rule objects are enabled by default until explicitly disabled by either a *DisableHubRule* or *ToggleHubRule* Action object.

The table below describes the available options and their valid value ranges.



Name:

Hub Rule Name: 00-AtSystemStart

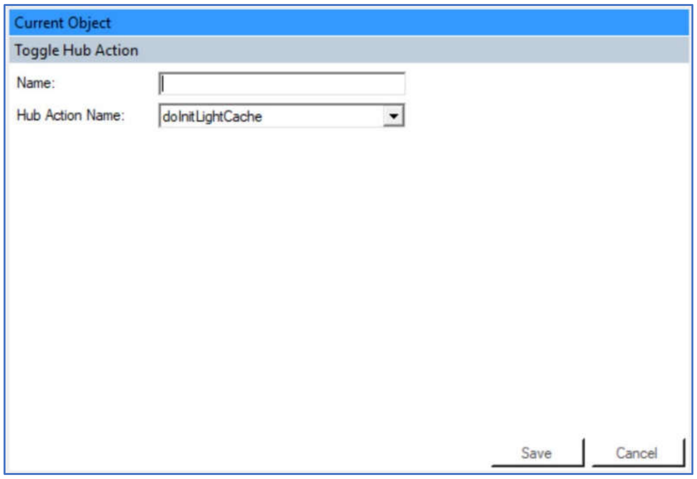
Figure 68: The Disable Hub Rule Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Hub Rule Name	This drop-down will present a list of all previously defined Rule objects.	Any drop-down list entry.

4.4.40 ToggleHubAction Action object

When executed, the *ToggleHubAction* Action object will toggle the enabled/disabled state of a previously defined Action object. When disabled, an Action object will no longer be executed regardless of whether a refering Rule object's conditional logic is satisfied or not. Note that all action objects are enabled by default until explicitly disabled by either a *DisableHubAction* or *ToggleHubAction* Action object.

The table below describes the available options and their valid value ranges.



Name:

Hub Action Name: doInitLightCache

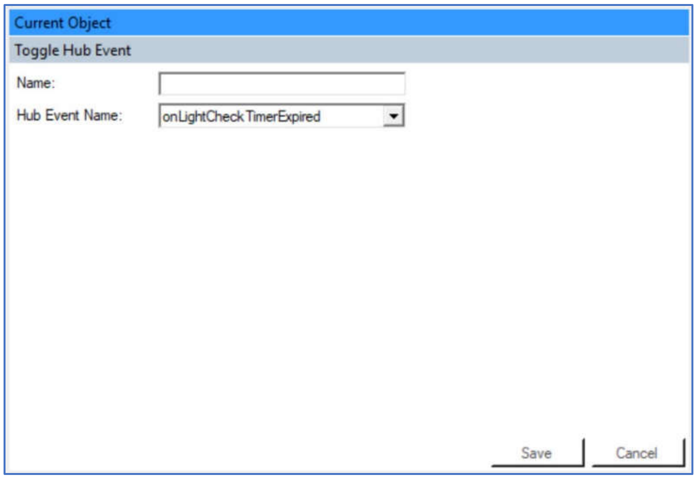
Figure 69: The Toggle Hub Action Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Hub Action Name	This drop-down will present a list of all previously defined Action objects excluding the *EnableHubAction*, *DisableHubAction*, and *ToggleHubAction* Action objects themselves.	Any drop-down list entry.

4.4.41 ToggleHubEvent Action object

When executed, the *ToggleHubEvent* Action object will toggle the enabled/disabled state of a previously defined Event object. When disabled, Event objects will still be queued into the rule engine's event queue but will be immediately discarded when received (an no associated Rule objects will be evaluated). Note that all event objects are enabled by default until explicitly disabled by either a *DisableHubEvent* or *ToggleHubEvent* Action object.

The table below describes the available options and their valid value ranges.



Name:

Hub Event Name: onLightCheckTimerExpired

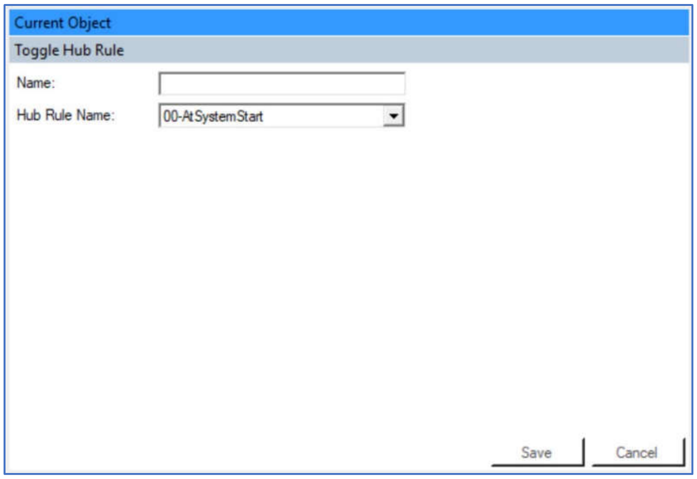
Figure 70: The Toggle Hub Event Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Hub Event Name	This drop-down will present a list of all previously defined Event objects.	Any drop-down list entry.

4.4.42 ToggleHubRule Action object

When executed, the *ToggleHubRule* Action object will toggle the enabled/disabled state of a previously defined Rule object. When disabled, a Rule object will no longer be evaluated/executed when any associated Event object is received by the rule engine. Note that all rule objects are enabled by default until explicitly disabled by either a *DisableHubRule* or *ToggleHubRule* Action object.

The table below describes the available options and their valid value ranges.



Name:

Hub Rule Name: 00-AtSystemStart

Figure 71: The Toggle Hub Rule Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Hub Rule Name	This drop-down will present a list of all previously defined Rule objects.	Any drop-down list entry.

4.4.43 SoftHubStop Action object

When executed, the *SoftHubStop* Action object will terminate the SoftHub application. Note once shutdown the SoftHub application will need to be manually restarted to resume operation.

The table below describes the available options and their valid value ranges.

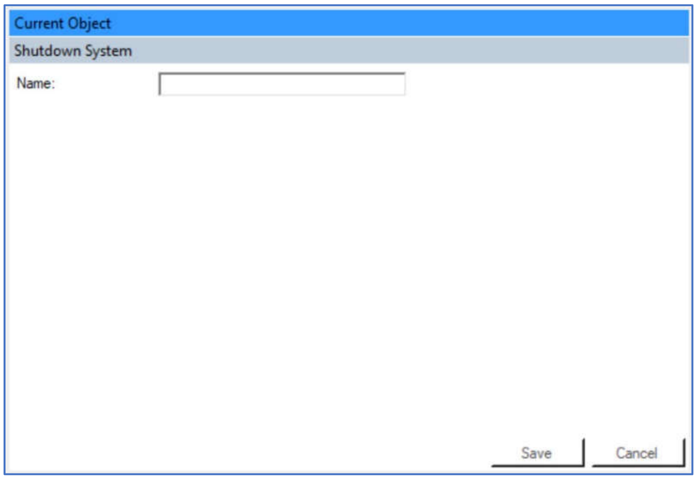


Figure 72: The SoftHub Stop Action object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.

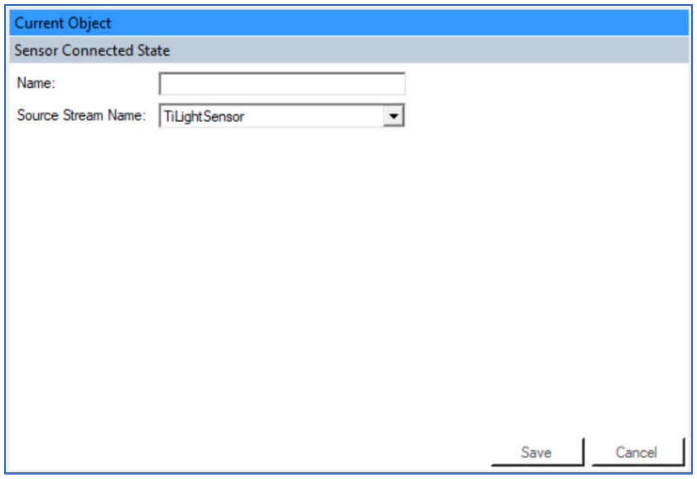
4.5 State Objects

This section will detail the individual *State Objects* currently supported by the SoftHub application. State objects are used to place conditional logic on the execution of individual Rule objects. When evaluated/executed, a State object will resolve to a logical "true" or "false" based on some current state within the SoftHub application. Examples would include whether a specific sensor value exceeds a user defined threshold or whether an input GPIO pin is currently "high". When defining the Rule objects themselves you will have the option to build complex Boolean logic statements comprised of one or more State objects to be evaluated. Boolean logic statements can be built using logical AND, OR, and NOT statements as well parenthesis to alter logical precedent.

4.5.01 SensorConnected State object

When evaluated, the *SensorConnected* State object will determine if the specified Sensor Stream object is currently connected to the SoftHub application. Note that there may be some lag when a connection is lost before the hardware reports the connection as permanently disconnected.

The table below describes the available options and their valid value ranges.


Current Object

Sensor Connected State

Name:

Source Stream Name: TiLightSensor

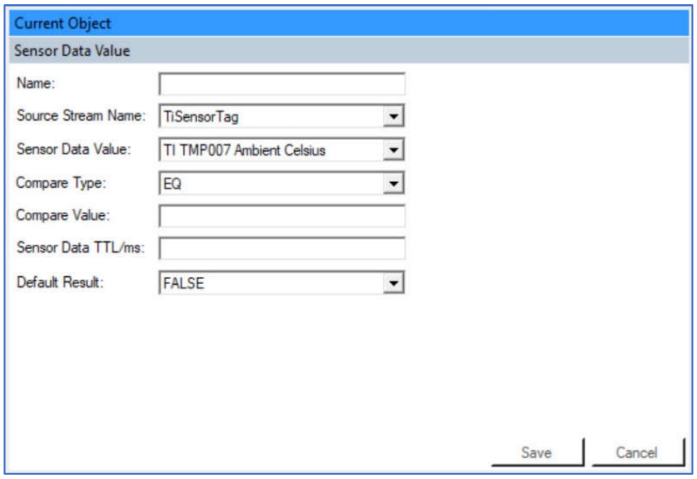
Figure 73: The Sensor Connected State object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Sensor Stream Name	This drop-down list will contain all of the previously defined Sensor Streams objects. The value of this state object will evaluate to *true* if the selected Source Stream object has been successfully connected to the system.	Any drop-down list entry.

4.5.02 SensorDataValue State object

When evaluated, the *SensorDataValue* State object will test the last received specified sensor value against a user defined threshold. Note that if the sensor has not yet connected or if it is connected but has not yet reported a packet, the value of this State object will be returned as indicated by the user specified “Default Result”.

The table below describes the available options and their valid value ranges.



Name:
Source Stream Name: TiSensorTag
Sensor Data Value: TI TMP007 Ambient Celsius
Compare Type: EQ
Compare Value:
Sensor Data TTL/ms:
Default Result: FALSE

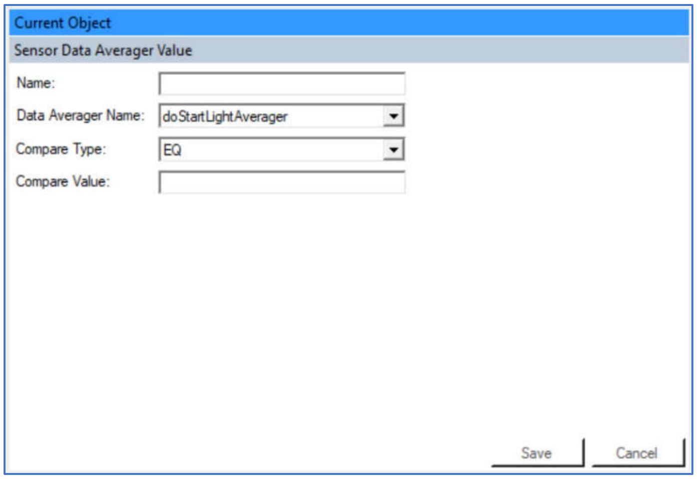
Figure 74: The Sensor Data Value State object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Sensor Stream Name	This drop-down list will contain each of the previously defined Sensor Streams objects. Select the Sensor Stream object that contains the data value that you wish to compare against.	Any drop-down list entry.
Sensor Data Value	This drop-down list will contain each of the data value fields previously enabled for the selected Sensor Stream object. Select the data value field you wish to compare against.	Any drop-down list entry.
Compare Type	In this drop-down are a list of valid compare types for the data type being compared against. Select the compare type to be made between the sensor's current data value and the specified compare value (EQ, NEQ, GT, NGT, LT, NLT).	Any drop-down list entry.
Compare Value	This is the value you wish to compare against the selected data item's current value.	Note that the valid range for this value will be type dependent. For example if the data type is a 16 bit unsigned integer the valid range will a numeric between 0 and 65,535.
Sensor Data TTL/ms	This value provides a time limit (in milliseconds) on the sensor data being tested against. If the sensor data is older than the time to live specified, the user selected default result will be returned. Note that a value of “0” indicates that the sensor value never expires.	A value between 0 to 4,294,967,295.
Default Result	This dropdown allows the user to select the default compare result to be returned when either the sensor data is unavailable or has expired per the value entered in the “Sensor Data TTL/ms” textbox.	Any drop-down list entry.

4.5.03 SensorDataAveragerValue State object

The *SensorDataAveragerValue* State object will compare the value currently available from the named averager object against a user defined threshold value.

The table below describes the available options and their valid value ranges.



Name:

Data Averager Name: doStartLightAverager

Compare Type: EQ

Compare Value:

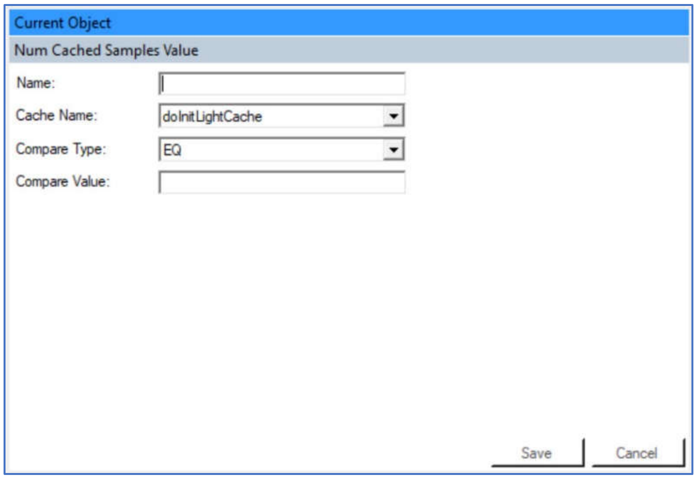
Figure 75: The Sensor Data Averager Value State object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Data Averager Name	This drop-down will list each of the previously defined Data Averager objects. Note that Data Averager objects are automatically derived any time a new *InitDataAverager* Action object is executed.	Any drop-down list entry.
Compare Type	In this drop-down you will find a list of logical compare type operators for comparing the counter's current value against the specified compare value (EQ, NEQ, GT, NGT, LT, NLT).	Any drop-down list entry.
Compare Value	This is the value you wish to compare against the selected counter object's current value.	This value must be a valid signed or unsigned floating point number.

4.5.04 NumCachedSamplesValue State object

The *NumCachedSamplesValue* State object will compare the number of cached samples available in the specified cache object against the user specified threshold value.

The table below describes the available options and their valid value ranges.



Name:

Cache Name: doInitLightCache

Compare Type: EQ

Compare Value:

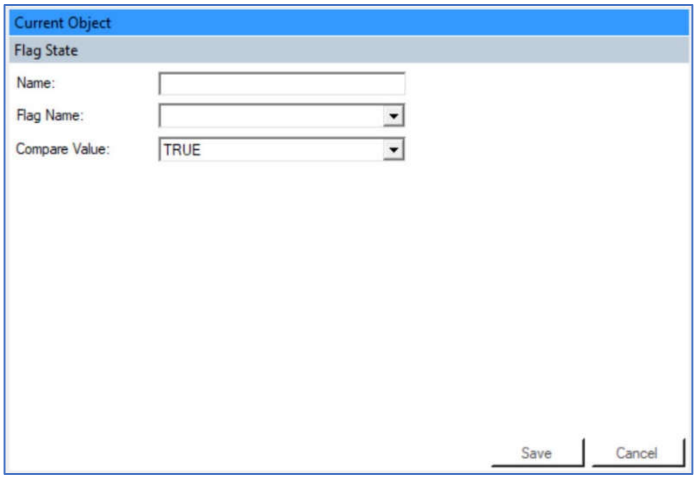
Figure 76: The Num Cached Samples Value State object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Cache Name	This drop-down will list each of the previously defined Sensor Data Cache objects. Note that Sensor Data Cache objects are automatically derived any time a new *InitDataCache* Action object is executed.	Any drop-down list entry.
Compare Type	This is the type of logical compare that will be performed between the actual number of cached samples and the specified Compare Value (EQ, NEQ, GT, NGT, LT, NLT).	Any drop-down list entry.
Compare Value	This is the value that will be compared against the actual number of cached values in the selected data cache.	Valid compare values are numeric values between 0 and 1,024.

4.5.05 FlagState State object

When evaluated, the *FlagState* State object will compare the value of the specified Flag object against the test value provided by the user.

The table below describes the available options and their valid value ranges.



Name:

Flag Name:

Compare Value: TRUE

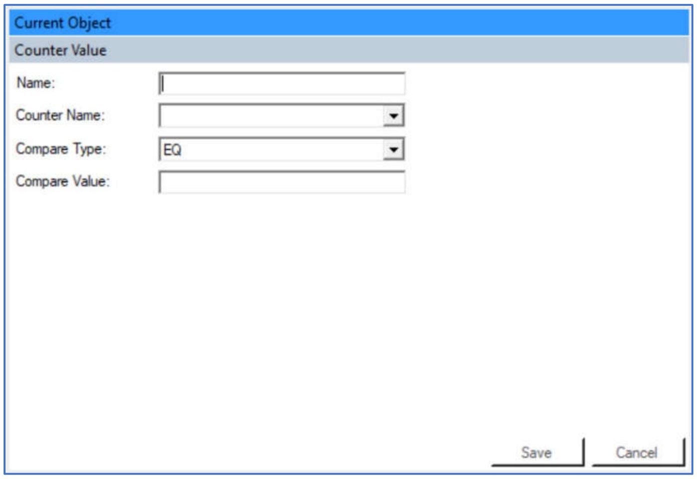
Figure 77: The Flag State State object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Flag Name	This drop-down will list each of the previously defined Flag objects. Note that Flag objects are automatically derived any time a new *InitializeFlag* Action object is executed.	Any drop-down list entry.
Compare Value	This is the value you wish to compare against the selected flag object's current value.	Any drop-down list entry.

4.5.06 CounterValue State object

When evaluated, the *CounterValue* State object will test the specified counter objects' current value against the user specified threshold value.

The table below describes the available options and their valid value ranges.



Name:

Counter Name:

Compare Type: EQ

Compare Value:

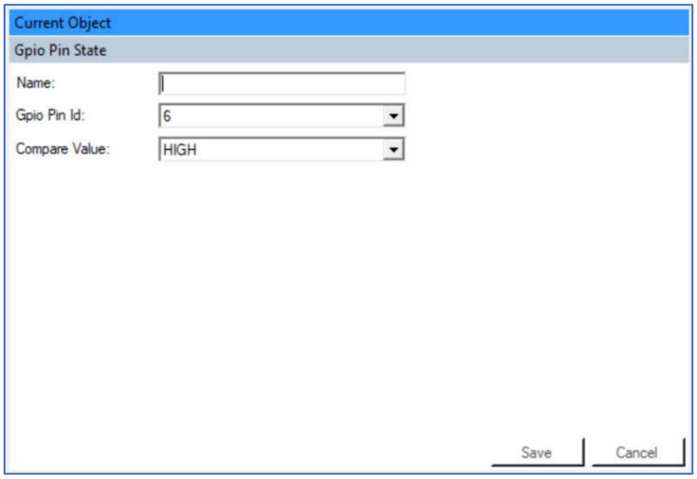
Figure 78: The Counter Value State object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Counter Name	This drop-down will list each of the previously defined Counter objects. Note that Counter objects are automatically derived any time a new *InitializeCounter* Action object is executed.	Any drop-down list entry.
Compare Type	This is the type of logical compare that will be performed between the actual counter value and the specified Compare Value (EQ, NEQ, GT, NGT, LT, NLT).	Any drop-down list entry.
Compare Value	This is the value you wish to compare against the selected counter object's current value.	This value must be a number between –9,223,372,036,854,775,808 and 9,223,372,036,854,775,807.

4.5.07 GpioPinState State object

When evaluated, the *GpioPinState* State object will test the specified GpioPinState against the user defined compare value.

The table below describes the available options and their valid value ranges.



Name:
Gpio Pin Id: 6
Compare Value: HIGH

Figure 79: The GPIO Pin State State object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
GPIO Pin Id	This drop-down lists the GPIO Input Pins that have previously been enabled in the System Options configuration screen.	Any drop-down list entry.
Compare Value	This is the value that will be compared against the selected GPIO pin's current value. The state object will evaluate to true if the compare value matches the GPIO pen's actual value, otherwise the state object will evaluate to false.	Any drop-down list entry.

4.5.08 TimeOfDay State object

When evaluated, the *TimeOfDay* State object will test the current time of day against the user specified compare value.

The table below describes the available options and their valid value ranges.

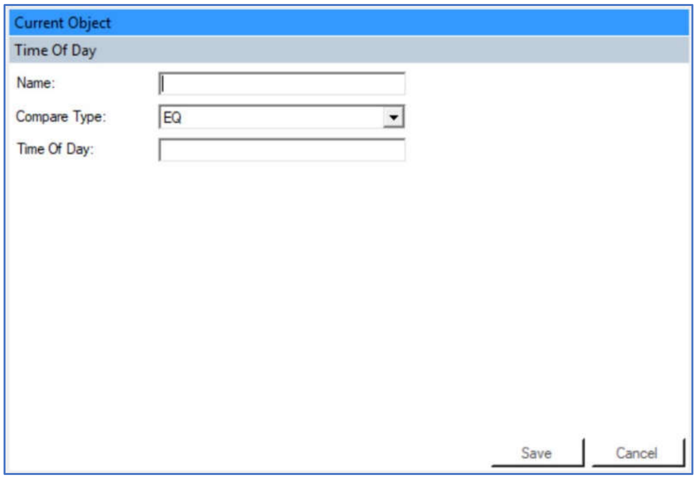


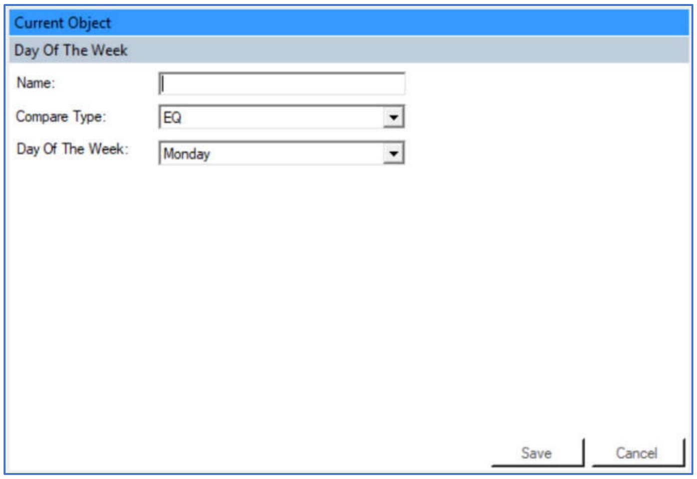
Figure 80: The Time Of The Day State object

Field Name	Description	Valid Values
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Compare Type	This is the type of logical compare that will be performed between the actual time of day and the specified compare value (EQ, NEQ, GT, NGT, LT, NLT).	Any drop-down list entry.
Time Of Day	This is the value you wish to compare against the actual current time of day.	Valid values must be specified in the 24 hour format of "HH:MM:SS" (e.g. "14:05:00" for 2:05 PM).

4.5.09 DayOfTheWeek State object

When evaluated, the *DayOfTheWeek* State object will test the current day of the week against the user specified compare value.

The table below describes the available options and their valid value ranges.



Name:

Compare Type: EQ

Day Of The Week: Monday

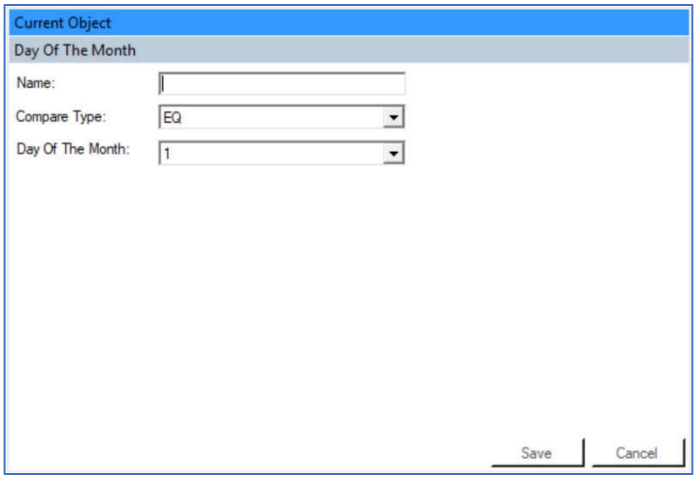
Figure 81: The Day Of The Week State object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Compare Type	This is the type of logical compare that will be performed between the actual day of the week and the specified compare value (EQ, NEQ, GT, NGT, LT, NLT).	Any drop-down list entry.
Day of the Week	This is a drop-down of day of the week values you wish to compare against the actual current day of the week.	Any drop-down list entry.

4.5.10 DayOfTheMonth State object

When evaluated, the *DayOfTheMonth* State object will test the current day of the month against the user specified compare value.

The table below describes the available options and their valid value ranges.



Name:

Compare Type: EQ

Day Of The Month: 1

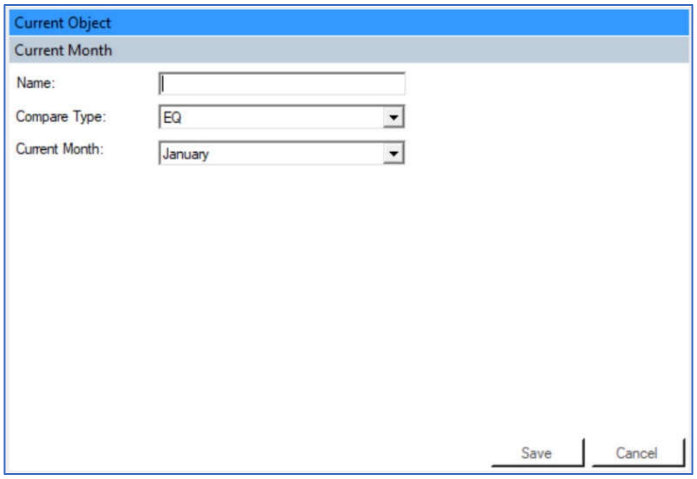
Figure 82: The Day Of The Month State object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Compare Type	This is the type of logical compare that will be performed between the actual day of the month and the specified compare value (EQ, NEQ, GT, NGT, LT, NLT).	Any drop-down list entry.
Day of the Month	This is a drop-down of day of the month values you wish to compare against the actual current day of the month.	Any drop-down list entry.

4.5.11 CurrentMonth State object

When evaluated, the *CurrentMonth* State object will test the current month against the user specified compare value.

The table below describes the available options and their valid value ranges.



Current Object

Current Month

Name:

Compare Type: EQ

Current Month: January

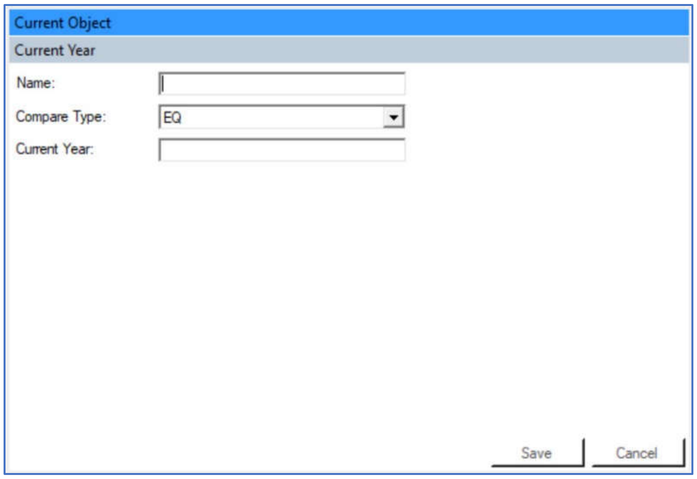
Figure 83: The Current Month State object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Compare Type	This is the type of logical compare that will be performed between the actual current month and the specified compare value (EQ, NEQ, GT, NGT, LT, NLT).	Any drop-down list entry.
Compare Month	In this drop-down you can select the month value you wish to test the current month value against.	Any drop-down list entry.

4.5.12 CurrentYear State object

When evaluated, the *CurrentYear* State object will test the current year against the user specified compare value.

The table below describes the available options and their valid value ranges.



Name:

Compare Type: EQ

Current Year:

Figure 84: The Current Year State object

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Compare Type	This is the type of logical compare that will be performed between the actual current year and the specified compare value (EQ, NEQ, GT, NGT, LT, NLT).	Any drop-down list entry.
Compare Year	This is the value you wish to compare against the actual current year.	Valid values are numeric values between 2016 and 9999.

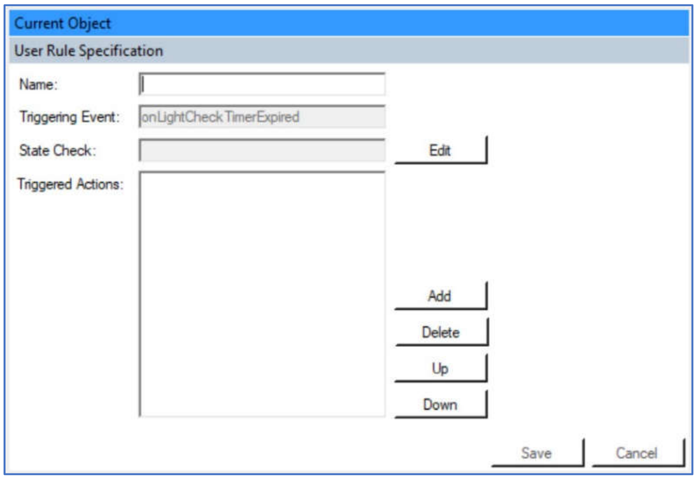
4.6 Rule Objects

Rule Objects ultimately control all aspects to the SoftHub behavior. Each Rule object is associated with a single Event object (though an Event object may be associated with multiple Rule objects). As events are detected by the various SoftHub application components, a corresponding "Event Object" is put into the rule engine's event queue. As the rule engine receives these events, it will check to see if the user has defined any rules to be evaluated for the corresponding event. If so, each rule found will be sequentially evaluated in the order listed by the user when building the configuration file.

"Evaluating" an individual rule consists of two steps, (1) evaluating any optional conditional logic the user has defined and, (2) if the conditionals are met (or there are none defined), executing the list of one or more associated Action objects (again, in the order as defined by the user).

4.6.01 UserRule Rule object (Main)

As noted above building a Rule object consists of two steps. To add any optional conditional logic, click the "Edit" button to the right of the “State Check” text-box to enter the State Check dialog screen. Once you are satisfied with your conditional logic, click the "Add" button next to the "Triggered Actions" list-box to build a list of Action objects to be executed when the rule’s conditional logic has been satisfied. As you add Action Objects to the list-box you can use the "Up", "Down", and "Delete" buttons to ensure that the selected Action objects are executed in the proper order.



Triggering Event: onLightCheckTimerExpired

Figure 85: The User Rule Specification Rule object

The table below describes the available options and their valid value ranges.

Field Name	**Description**	**Valid Values**
Name	This is the name by which you will refer to this object entry when cross-referenced by other configuration objects. The name you choose should appropriately describe the object you are creating.	Names may be between 4 to 40 characters in length and may include alphanumeric characters as well as dashes, underscores, and periods only.
Triggering Event	This text-box will identify the Event object which will trigger the evaluation of the Rule object being created. Note that his textbox is un-editable.	N/A

State Check	Using this edit box you can optionally create complex Boolean logic statements to be evaluated to conditionally execute the rule object’s Action object list. At run time, if the evaluation of the resulting logic statement resolves to *true*, the rule’s Action objects will be executed. If the resulting logic statement resolves to *false*, no further processing of the current rule is undertaken. Use the "Edit" button to enter your Boolean logic statements in the separate *State Check* definition screen.	N/A
Triggered Actions	In this edit box you can add, edit, and arrange the action objects you wish the rule to execute. Clicking the *Add* button will open a separate *Triggered Actions* definition screen to select your action objects. You can then use the *Delete*, *Up*, and *Down* buttons to set the order in which the actions are to be performed.	N/A

4.6.02 UserRule Rule object (State Check)

The *State Check* screen is used to create any optional conditional logic to be evaluated each time the associated rule is triggered. If when evaluated the logic statement resolves to "true" then the Rule object's Action object list will be executed. If however the logic statement resolves to "false" then the Rule object being evaluated will not be further processed. Note that the buttons to the right can be used to logically connect one or more State objects to create more complex conditional logic.

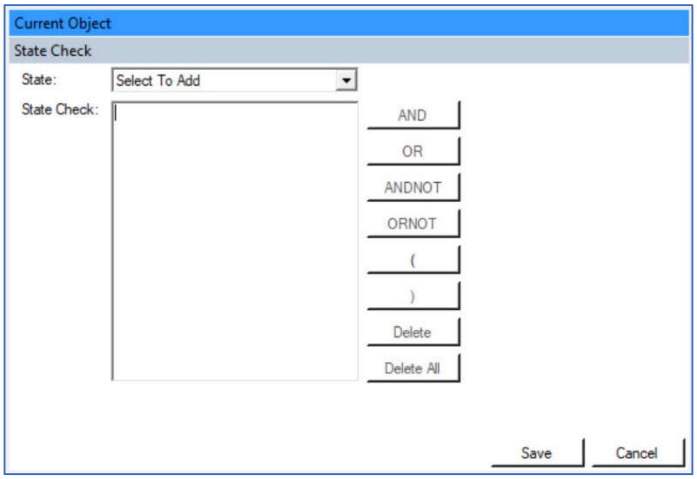


Figure 86: The Rule Object State Check dialog

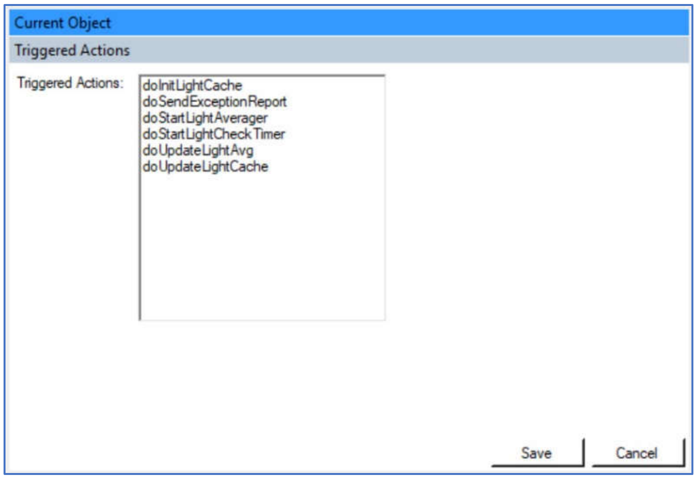
The table below describes the available options and their valid value ranges.

Field Name	**Description**	**Valid Values**
State	This drop-down contains a list of all state objects defined in the current rule set. Use this dropdown to add one or more State Objects to your conditional logic statement.	Any drop-down list entry
State Check	Use the state check edit box and the edit buttons to the right of the list-box to edit and build your logical conditional statement as required.	N/A

4.6.03 UserRule Rule object (Triggered Actions)

The *Triggered Actions* screen allows the user to build a list of one or more Action objects to be executed when the associated rule has been triggered and any optional state logic has been successfully evaluated. To select one or more Action Objects, multiselect the entry(s) and click the *Save* button. Note that the order in which the actions are executed can be altered in the Rule object's main screen.

The table below describes the available options and their valid value ranges.



doInitLightCache
doSendExceptionReport
doStartLightAverager
doStartLightCheckTimer
doUpdateLightAvg
doUpdateLightCache

Figure 87: The Rule Object Triggered Actions dialog

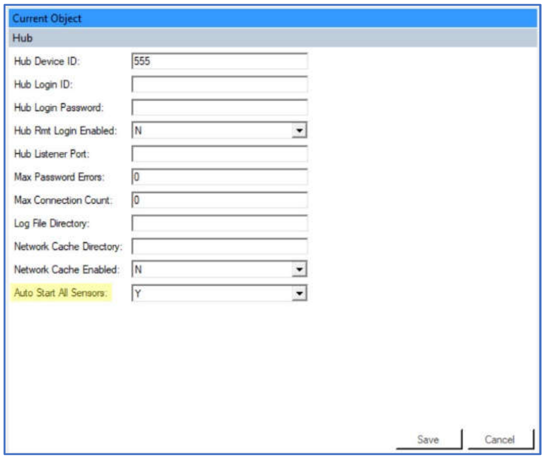
Field Name	**Description**	**Valid Values**
Triggered Actions	In this edit box you will see a list of all the currently defined action objects for the current rule set. Select action objects to be added to the rule object by highlighting the and clicking the save button.	Any drop-down list entry (multi-select)

5. Example Rulesets

This section presents five simplistic example scenarios used to illustrate some of the more common SoftHub functionality. Each example will provide a brief functional summary followed by an in-detail description of each of the actual objects used to implement the scenario ruleset. At the end of each example is a table that describes the presented ruleset as a series of *When/If/Do* statements. When creating your own rulesets these types of tables should be used as a starting point to map out the objects that will be required.

Note that for continuity individual object definitions are presented in the order listed in each scenario's configuration object tree (i.e. alphabetically). Since objects will often cross reference each other there will be instances in the object descriptions where, if created in the specific order listed, would not yet exist. In particular all Rule objects themselves, while presented first, are in practice generally created last.

Also note that for all scenario's presented it is assumed that the Hub System Options setting Auto Start All Sensors has been enabled (set to "Y"). For simplicity, it is assumed that the sensors used in the example will remain connected at all times. Failure to change this setting will make any of the examples non-operational.



Hub Device ID:	555
Hub Login ID:	
Hub Login Password:	
Hub Rmt Login Enabled:	N
Hub Listener Port:	
Max Password Errors:	0
Max Connection Count:	0
Log File Directory:	
Network Cache Directory:	
Network Cache Enabled:	N
Auto Start All Sensors:	Y

Figure 88: The Auto Start All Sensors system setting

5.1 Simple Data Forwarding

Example 1 presents a simple, though very common, configuration file to implement the following:

- On application startup it will implicitly connect to two sensors (via the *Auto Start All Sensors* System Option), (1) an ANT Garmin Tempe sensor, and (2) a BLE TI SensorTag sensor.
- Once connected, every packet received from either sensor will be forwarded to a remote packet capture server.

Figure 89 illustrates the object tree view of the ruleset used to implement this scenario. Briefly, the objects and the purpose of each are also summarized. The following sections will then describe each object and its settings in detail.

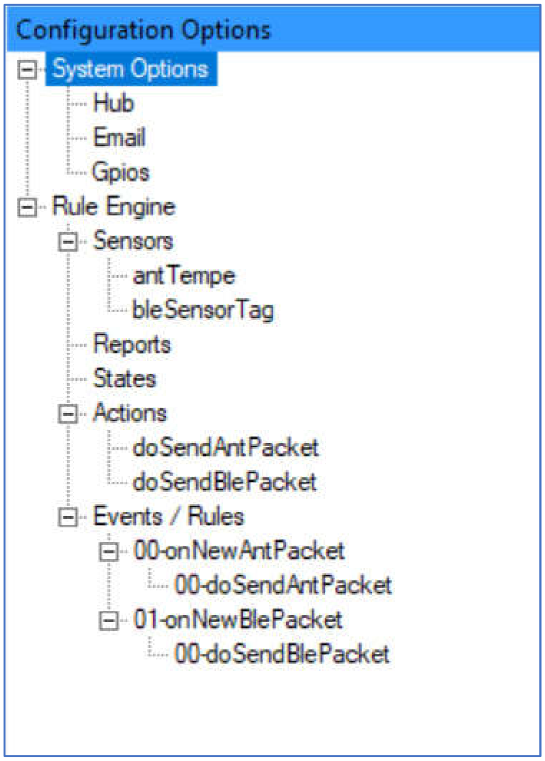


Figure 89: Example 1 configuration object tree

Sensor Stream objects:

- **antTempe** – Identifies the ANT sensor to connect to.
- **bleSensorTag** – Identifies the BLE sensor to connect to.

Action objects:

- **doSendAntPacket** – This *SendData* Action object will send the most recent packet from its associated Sensor Stream object (antTemp) to a user specified remote packet capture server.
- **doSendBlePacket** – This *SendData* Action object will send the most recent packet from its associated Sensor Stream object (bleSensorTag) to a user specified remote packet capture server.

Event objects:

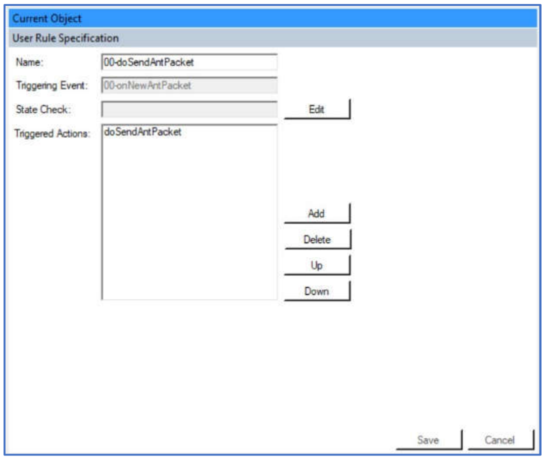
- **00-onNewAntPacket** – This *NewSensorPacket* Event object will be queued every time a new packet is received from its associated Sensor Stream object (antTempe).
- **01-onNewBlePacket** – This *NewSensorPacket* Event object will be queued every time a new packet is received from its associated Sensor Stream object (bleSensorTag).

Rule objects:

- **00-doSendAntPacket** – This Rule object will be evaluated every time its associated Event object (00-onNewAntPacket) is queued.
- **00-doSendBlePacket** – This Rule object will be evaluated every time its associated Event object (01-onNewBlePacket) is queued.

Full Rule object definition(s):

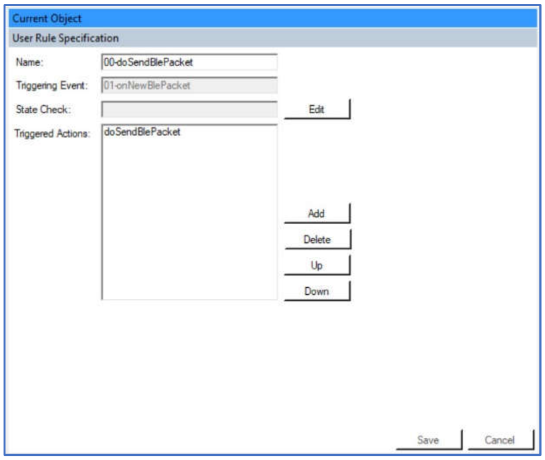
Figure 90 shows the full definition of the *00-doSendAntPacket* Rule object. It will be evaluated every time its triggering Event object, *00-onNewAntPacket*, gets read from the rule engine’s event queue. It has no conditional State Check logic to evaluate so its Action object(s) will always be executed. When executed its single Action object, *doSendAntPacket,* will send the most recently received packet from the ANT Tempe sensor to a user specified remote packet capture server.



Name: 00-doSendAntPacket
Triggering Event: 00-onNewAntPacket
State Check:
Triggered Actions: doSendAntPacket

Figure 90: The 00-doSendAntPacket Rule object definition

Figure 91 shows the full definition of the *00-doSendBlePacket* Rule object. It will be evaluated every time its triggering Event object, *01-onNewBlePacket*, gets read from the rule engine’s event queue. It has no conditional State Check logic to evaluate so its Action object(s) will always be executed. When executed its single Action object, *doSendBlePacket,* will send the most recently received packet from the BLE SensorTag sensor to a user specified remote packet capture server.



Name: 00-doSendBlePacket
Triggering Event: 01-onNewBlePacket
State Check:
Triggered Actions: doSendBlePacket

Figure 91: The 00-doSendBlePacket Rule object definition

Full Sensor Stream object definition(s):

Figure 92 shows the full definition for the *antTempe* ANT Sensor Stream object. From the dropdown of supported ANT sensors a *Sensor Type* of “Ant Environment” was selected (the ANT profile for the Garmin Tempe). No specific *Sensor ID* was defined so that the SoftHub will connect to the first Tempe sensor available. Lastly, when originally creating the antTempe object a *New Packet Event* Event object was specified, *00-OnNewAntPacket*. This instructs the SoftHub to queue the named event every time a new packet is received from the Tempe sensor that connects to the SoftHub via this specific Sensor Stream definition.

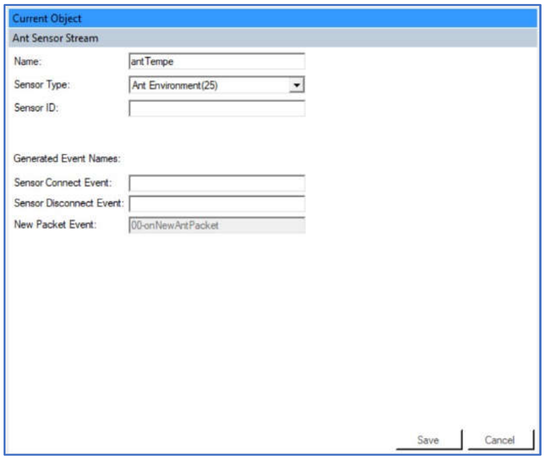

Current Object
Ant Sensor Stream
Name: antTempe
Sensor Type: Ant Environment(25)
Sensor ID:
Generated Event Names:
Sensor Connect Event:
Sensor Disconnect Event:
New Packet Event: 00-onNewAntPacket

Figure 92: The antTempe Sensor Stream object definition

Figure 93 shows the full definition for the *bleSensorTag* BLE Sensor Stream object. From the dropdown of supported BLE sensors a *Sensor Type of* “TI CC2650 SendorTag” was selected. No specific *Sensor ID* was defined so that the SoftHub will connect to the first SensorTag sensor available. Unlike ANT sensors which will always send all sensor data available, BLE sensors must have each sensor data value enabled individually. For this example, only the data from the Light Intensity sensor has been enabled from the list of values the sensor supports. Note however that this is a multi-select list-box and one or more values may be selected as required.

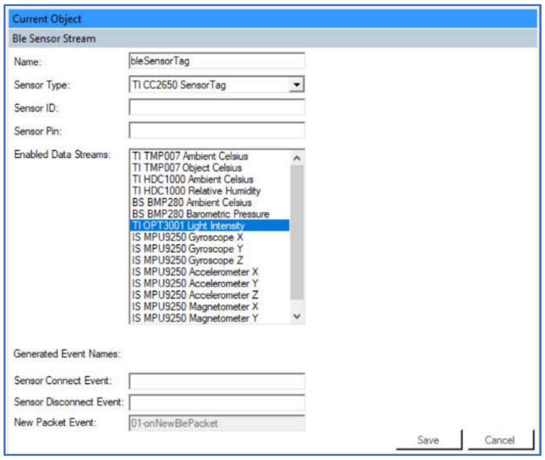

Current Object
Ble Sensor Stream
Name: bleSensorTag
Sensor Type: TI CC2650 SensorTag
Sensor ID:
Sensor Pin:
Enabled Data Streams:
TI TMP007 Ambient Celsius
TI TMP007 Object Celsius
TI HDC1000 Ambient Celsius
TI HDC1000 Relative Humidity
BS BMP280 Ambient Celsius
BS BMP280 Barometric Pressure
TI OPT3001 Light Intensity
IS MPU9250 Gyroscope X
IS MPU9250 Gyroscope Y
IS MPU9250 Gyroscope Z
IS MPU9250 Accelerometer X
IS MPU9250 Accelerometer Y
IS MPU9250 Accelerometer Z
IS MPU9250 Magnetometer X
IS MPU9250 Magnetometer Y
Generated Event Names:
Sensor Connect Event:
Sensor Disconnect Event:
New Packet Event: 01-onNewBlePacket

Figure 93: The bleSensorTag Sensor Stream object definition

Lastly, when originally creating the bleSensorTag object a *New Packet Event* Event object was specified, *01-OnNewBlePacket*. This instructs the SoftHub to queue the named event every time a new packet is received from the SensorTag sensor that connects to the SoftHub via this specific Sensor Stream definition.

<u>Full Action object definition(s):</u>

Figure 94 shows the full definition for the *doSendAntPacket* SendData Action object. When executed this object will send the latest packet data from the chosen Sensor Stream to the user specified IP and port address over a TLS encrypted connection.

When defining a SendData object, the Source Stream *Name* dropdown will provide a list of all Sensor Streams the user has previously defined. For this object the *antTempe* Sensor Stream object was selected. The *Destination IP Address* identifies the IP address of a remote packet capture server to forward the packets to (typically a server running the OmnIoT Packet Capture Service). The *Destination Port* identifies what port the packet capture service has been configured to listen on.

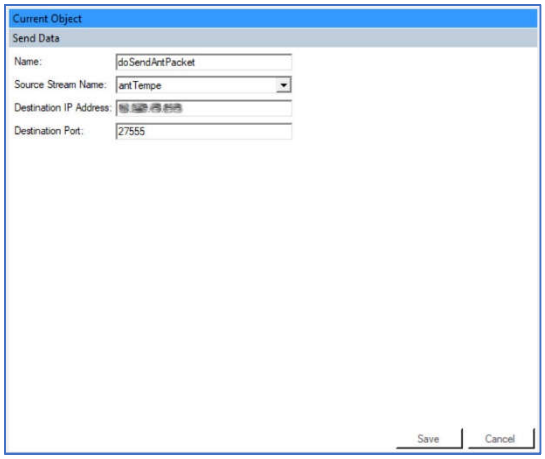

Current Object
Send Data
Name: doSendAntPacket
Source Stream Name: antTempe
Destination IP Address: [redacted]
Destination Port: 27555

Figure 94: The doSendAntPacket Action object definition

Figure 95 shows the full definition for the *doSendBlePacket* SendData Action object. When executed this object will send the latest packet data from the chosen Sensor Stream to the user specified IP and port address over a TLS encrypted connection.

When defining a SendData object, the Source Stream *Name* dropdown will provide a list of all Sensor Streams the user has previously defined. For this object the *bleSensorTag* Sensor Stream object was selected. The *Destination IP Address* identifies the IP address of a remote packet capture server to forward the packets to (typically a server running the OmnIoT Packet Capture Service). The *Destination Port* identifies what port the packet capture service has been configured to listen on.

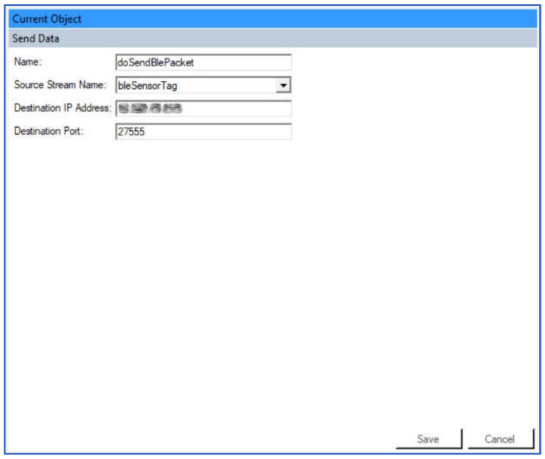

Current Object
Send Data
Name: doSendBlePacket
Source Stream Name: bleSensorTag
Destination IP Address: [redacted]
Destination Port: 27555

Figure 95: The doSendBlePacket Action object definition

Example Summary:

Example 1 portrayed one of the simplest, yet most common, IoT application scenarios. SoftHub rulesets can be thought of as a series of *when/if/do* statements (i.e. **WHEN** *some event occurs*, **IF** *some conditions are met*, **DO** *some action(s)*). The table below provides a summary of how the SoftHub's rule engine will process the ruleset described in example 1:

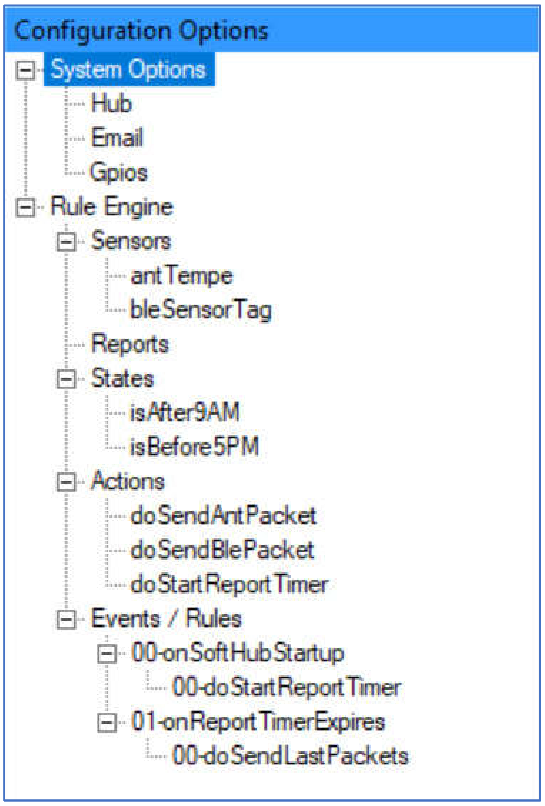
Event (When)	**Rule**	**State (If)**	**Action (Do)**
00-onNewAntPacket: A new packet has been received from the connected Tempe sensor.	00-doSendAntPacket	N/A, Do always	doSendAntPacket: Send the most recent Tempe packet to a remote packet capture server.
01-onNewBlePacket: A new packet has been received from the connected SensorTag sensor.	00-doSendBlePacket	N/A, Do always	doSendBlePacket: Send the most recent SensorTag packet to a remote packet capture server.

5.2 Timers and Conditionals

Example 2 illustrates a variation on example 1 and introduces Timer objects as well as State objects to conditionally control the execution of Rule objects. Example 2’s configuration file will do the following:

- On application startup it will connect to two sensors (via the *Auto Start All Sensors* System Option), (1) an ANT Garmin Tempe sensor, and (2) a BLE TI SensorTag sensor.
- Also on startup it will create and start a Continuous Timer object which will emit Timer Expired events every 30 seconds.
- During the hours of 9 AM and 5PM, every time the timer expired event occurs (i.e. every 30 seconds) the most recent packet received from each of the connected sensors will be forwarded to a remote packet capture server.

Figure 96 depicts the object tree view of the ruleset used to implement this scenario. Briefly, the objects and the purpose of each are also summarized. The following sections will then describe each object and its settings in further detail.



Configuration Options

- System Options
  - Hub
  - Email
  - Gpios
- Rule Engine
  - Sensors
    - antTempe
    - bleSensorTag
  - Reports
  - States
    - isAfter9AM
    - isBefore5PM
  - Actions
    - doSendAntPacket
    - doSendBlePacket
    - doStartReportTimer
  - Events / Rules
    - 00-onSoftHubStartup
      - 00-doStartReportTimer
    - 01-onReportTimerExpires
      - 00-doSendLastPackets

Figure 96: Example 2 configuration object tree

Sensor Stream objects:

- **antTempe** – Identifies the ANT sensor to connect to.
- **bleSensorTag** – Identifies the BLE sensor to connect to.

State objects:

- **isAfter9AM** – This *TimeOfDay* State object will be true when executed if the current time of day is after 9 AM.
- **isBefore5PM** – This *TimeOfDay* State object will be true when executed if the current time of day is before 5 PM.

Action objects:

- **doSendAntPacket** – This *SendData* Action object will send the most recent packet from its associated Sensor Stream object (antTemp) to a user specified remote packet capture server.
- **doSendBlePacket** – This *SendData* Action object will send the most recent packet from its associated Sensor Stream object (bleSensorTag) to a user specified remote packet capture server.
- **doStartReportTimer** – This *StartContinuousTimer* Action object will create and start a continuously repeating timer that will expire every 30 seconds.

Event objects:

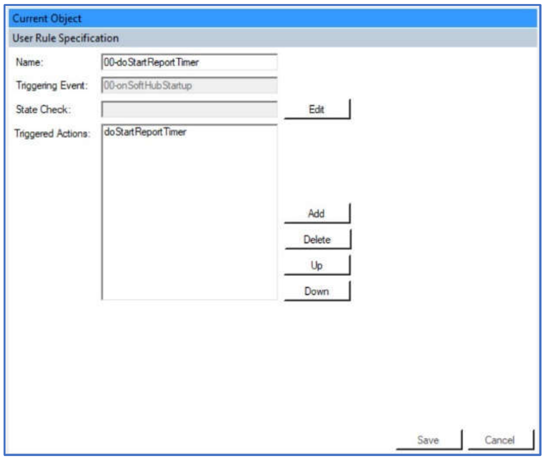
- **00-onSoftHubStartup** – When the SoftHub application starts, this *AppStarting* Event object will be queued as the first event object to be processed by the rule engine.
- **01-onReportTimerExpires** – This *TimerExpired* Event object will be queued every 30 seconds as its associated timer object, created via the *doStartReportTimer* Action object, expires.

Rule objects:

- **00-doStartReportTimer –** This Rule object will be evaluated every time its associated Event object (00-onSoftHubStartup) is queued.
- **00-doSendLastPackets –** This Rule object will be evaluated every time its associated Event object (01-onReportTimerExpires) is queued.

Full Rule object definition(s):

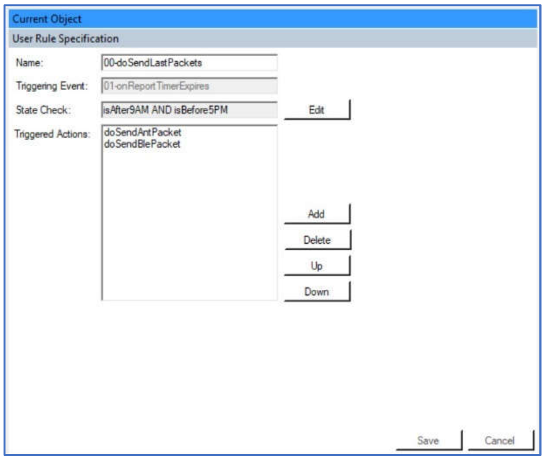
Figure 97 shows the full definition of the *00-doStartReportTimer* Rule object. It will be evaluated before any other rule when the initial *00-onSoftHubStartup* (AppStarting) Event object gets queued. It has no conditional State Check logic to evaluate so its Action object(s) will always be executed. When executed its single Action object, *doStartReportTimer,* will create and start a continuous report timer which will generate and queue an 00-onReportTimerExpires (*TimerExpired*) Event object each time it expires (every 30 seconds).



Name: 00-doStartReportTimer
Triggering Event: 00-onSoftHubStartup
State Check:
Triggered Actions: doStartReportTimer

Figure 97: The 00-doStartReportTimer Rule object definition

Figure 98 shows the full definition of the *00-doSendLastPackets* Rule object. This rule will be evaluated every time the *01-onReportTimerExpires* (TimerExired) Event object is queued. This rule has State Check conditional logic associated with it which will be evaluated prior to any Action objects being executed. In this case the two State objects (*isAfter9AM* and *isBefore5PM*) will be executed and their return values logically AND'ed together. Only if the result of the logical expression is *true* will the rule's two action objects be executed.



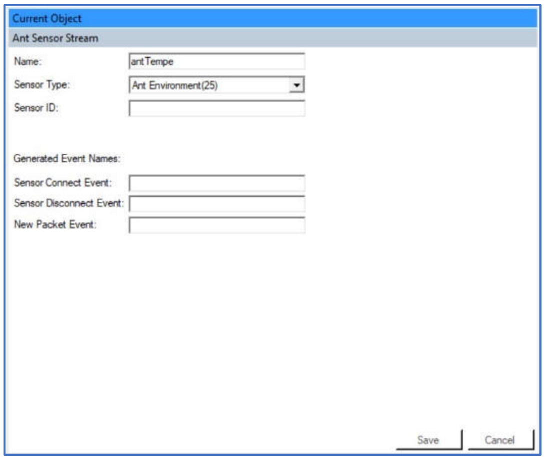
Name: 00-doSendLastPackets
Triggering Event: 01-onReportTimerExpires
State Check: isAfter9AM AND isBefore5PM
Triggered Actions: doSendAntPacket
doSendBlePacket

Figure 98: The 00-doSendLastPackets Rule object definition

As in the previous example, the two Action objects, *doSendAntPacket* and *doSendBlePacket*, will send the last received packets from their associated Sensor Stream objects to a remote packet capture server.

Full Sensor Stream object definition(s):

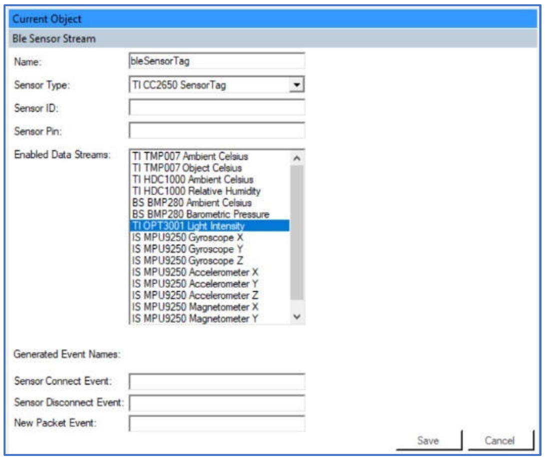
Figure 99 shows the full definition for the *antTempe* ANT Sensor Stream object. From the dropdown of supported ANT sensors a *Sensor Type* of “Ant Environment” was selected (the ANT profile for the Garmin Tempe). No specific *Sensor ID* was defined so that the SoftHub will connect to the first Tempe sensor available. Note that unlike the previous example the *New Packet Event* dialog box has been left blank as we do not require new packet event notifications for this scenario.



Name: antTempe
Sensor Type: Ant Environment(25)

Figure 99: The antTempe Sensor Stream object definition

Figure 100 shows the full definition for the *bleSensorTag* BLE Sensor Stream object. From the dropdown of supported BLE sensors a *Sensor Type of* “TI CC2650 SendorTag” was selected. No specific *Sensor ID* was defined so that the SoftHub will connect to the first SensorTag sensor available. Unlike ANT sensors which will always send all sensor data available, BLE sensors must generally have supported sensor data values enabled individually. For this example, only the data from the Light Intensity sensor has been enabled from the list of values the sensor supports. Note that this is a multi-select list-box and one or more values may be selected as required. Also note that unlike the example 1, the *New Packet Event* dialog box has been left blank as we do not require new packet event notifications for this scenario.

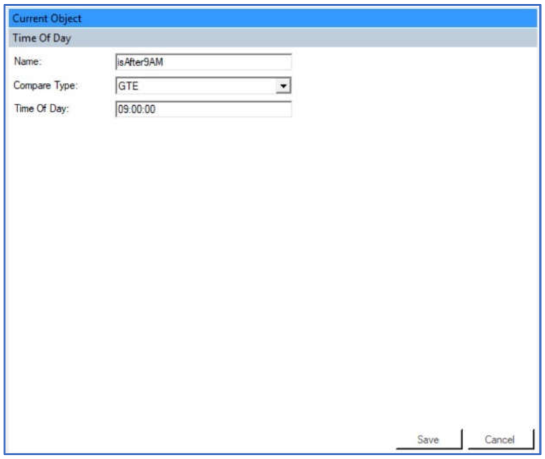


Name: bleSensorTag
Sensor Type: TI CC2650 SensorTag
Enabled Data Streams:
- TI TMP007 Ambient Celsius
- TI TMP007 Object Celsius
- TI HDC1000 Ambient Celsius
- TI HDC1000 Relative Humidity
- BS BMP280 Ambient Celsius
- BS BMP280 Barometric Pressure
- TI OPT3001 Light Intensity
- IS MPU9250 Gyroscope X
- IS MPU9250 Gyroscope Y
- IS MPU9250 Gyroscope Z
- IS MPU9250 Accelerometer X
- IS MPU9250 Accelerometer Y
- IS MPU9250 Accelerometer Z
- IS MPU9250 Magnetometer X
- IS MPU9250 Magnetometer Y

Figure 100: The bleSensorTag Sensor Stream object definition

Full State object definition(s):

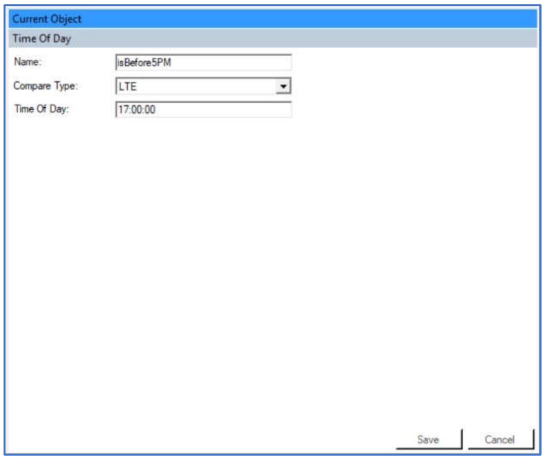
Figure 101 shows the full definition for the *isAfter9AM* TimeOfDay State object. When executed this object will return a value of either *true* or *false*. As defined in the example, if the current time of day is *greater than or equal to* (GTE) the user specified value in the *Time of Day* dialog box then this State object will be *true*, otherwise it will be *false*.



Name: isAfter9AM
Compare Type: GTE
Time Of Day: 09:00:00

Figure 101: The isAfter9AM State object definition

Figure 102 shows the full definition for the *isBefore5PM* TimeOfDay State object. When executed this object will return a value of either *true* or *false*. As defined in the example, if the current time of day is *less than or equal to* (LTE) the user specified value in the *Time of Day* dialog box then this State object will be *true*, otherwise it will be *false*.



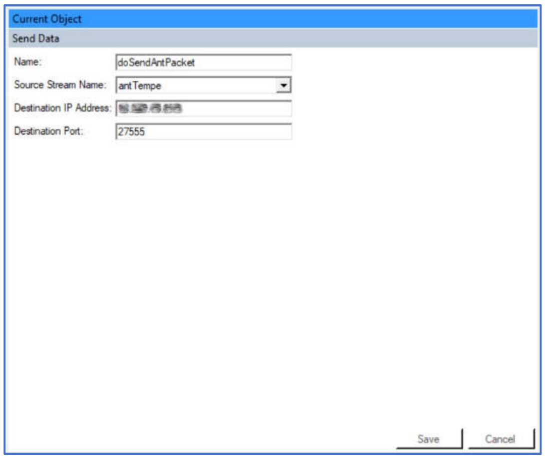
Name: isBefore5PM
Compare Type: LTE
Time Of Day: 17:00:00

Figure 102: The isBefore5PM State object definition

Full Action object definition(s):

Figure 103 shows the full definition for the *doSendAntPacket* SendData Action object. When executed this object will transmit the latest packet data from the chosen Sensor Stream to the user specified IP and port address over a TLS encrypted connection.

When defining a SendData object, the *Source Stream Name* dropdown will provide a list of all Sensor Streams the user has previously defined. For this object the *antTempe* Sensor Stream object was selected. The *Destination IP Address* identifies the IP address of a remote packet capture server to forward the packets to (typically a server running the OmnIoT Packet Capture Service). The *Destination Port* identifies what port the packet capture service has been configured to listen on.



Current Object

Send Data

Name: doSendAntPacket

Source Stream Name: antTempe

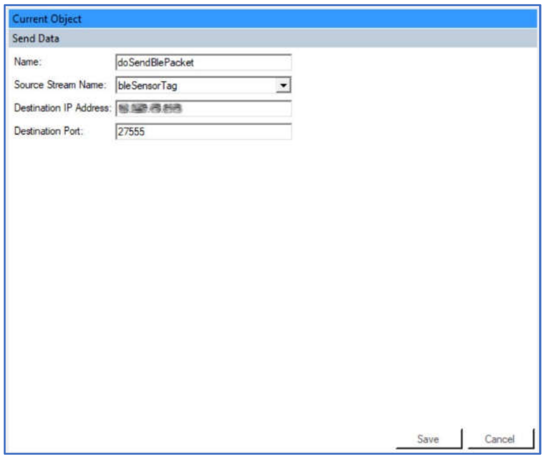
Destination IP Address: [redacted]

Destination Port: 27555

Figure 103: The doSendAntPacket Action object definition

Figure 104 shows the full definition for the *doSendBlePacket* SendData Action object. When executed this object will send the latest packet data from the chosen Sensor Stream to the user specified IP and port address over a TLS encrypted connection.

When defining a SendData object, the Source Stream *Name* dropdown will provide a list of all Sensor Streams the user has previously defined. For this object the *bleSensorTag* Sensor Stream object was selected. The *Destination IP Address* identifies the IP address of a remote packet capture server to forward the packets to (typically a server running the OmnIoT Packet Capture Service). The *Destination Port* identifies what port the packet capture service has been configured to listen on.



Current Object

Send Data

Name: doSendBlePacket

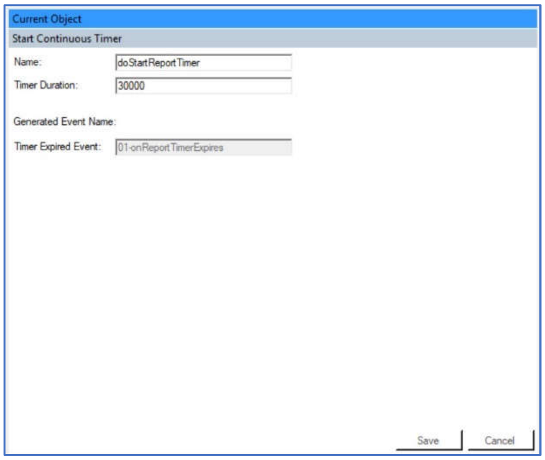
Source Stream Name: bleSensorTag

Destination IP Address: [redacted]

Destination Port: 27555

Figure 104: The doSendBlePacket Action object definition

Figure 105 shows the full definition for the *doStartReportTimer* StartContinuousTimer Action object. When executed this object will create and start a new continuous timer object. Continuous timer objects will continually restart themselves on expiration. For this example the *Timer Duration* has been set to 30000 millseconds (i.e 30 seconds). The *Timer Expired Event* dialog box defines the optional TimerExpired object to be queued each time the timer expires. Here, every 30 seconds an *01-onReportTimerExpires* TimerExpired object will be put to the rule engine's event queue.



Name: doStartReportTimer
Timer Duration: 30000
Generated Event Name:
Timer Expired Event: 01-onReportTimerExpires

Figure 37: The doStartReportTimer Action object definition

Example Summary:

Example 1 portrayed one of the simplest, yet most common, IoT application scenarios. Example 2 introduces Timer Events and Rule object conditionals (State objects). SoftHub rulesets can be thought of as a series of *when/if/do* statements (i.e. **WHEN** *some event occurs*, **IF** *some conditions are met*, **DO** *some action(s)*). The table below provides a summary of how the SoftHub's rule engine will process the ruleset described in Example 2:

Event (When)	**Rule**	**State (If)**	**Action (Do)**
00-onSoftHubStartup: The SoftHub is initially starting up.	00-doStartReportTimer	N/A, Do always	doStartReportTimer: Create and start a continuous timer object set to expire every 30 seconds.
00-onReportTimerExpires: A timer expired event object has been queued by the report timer.	00-doSendLastPackets	Do only if the isAfter9AM *AND* isBefore5PM State objects are both true.	doSendAntPacket: Send the most recent Tempe packet to a remote packet capture server.
As above	As above	As above	doSendBlePacket: Send the most recent SensorTag packet to a remote packet capture server.

5.3 Email and SMS Alerts

Example 3 extends the functionality of example 1 and introduces threshold based alarms and the *SendEmail* Action object. Example 3's configuration file will do the following (new functionality is noted in *italics*):

- On application startup it will implicitly connect to two sensors (via the *Auto Start All Sensors* System Option), (1) an ANT Garmin Tempe sensor, and (2) a BLE TI SensorTag sensor.
- Once connected, every packet received from either sensor will be forwarded to a remote packet capture server.
- *At any time, if the SensorTag light sensor data indicates that the lights are "off" (i.e. below our threshold value) an alert email will be sent*.
- *Once triggered, additional email alerts will be disabled for the following 15 minutes*.

Figure 106 illustrates the object tree view of the ruleset used to implement this scenario. Briefly, the objects and the purpose of each are also summarized. The following sections will then describe each object and its specific settings in further detail.

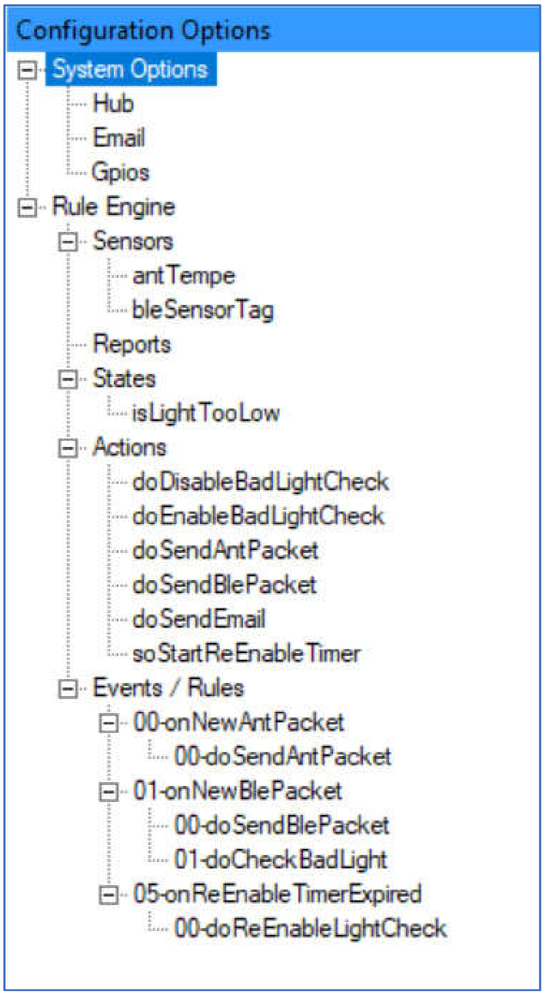


Figure 38: Example 3 configuration object tree

Sensor Stream objects:

- **antTempe** – Identifies the ANT sensor to connect to.
- **bleSensorTag** – Identifies the BLE sensor to connect to.

State objects:

- **isLightTooLow** – This *SensorDataValue* State object checks the light value from the bleSensorTag Sensor Stream against a user defined threshold.

Action objects:

- **doDisableBadLightCheck** – This *DisableHubRule* Action object will temporarily disable the 01-doCheckBadLight Rule object from being evaluated.
- **doEnableBadLightCheck** – This *EnableHubRule* Action object will Reenable the 01-doCheckBadLight Rule object for evaluation.
- **doSendAntPacket** – This *SendData* Action object will send the most recent packet from its associated Sensor Stream object (antTemp) to a user specified remote packet capture server.
- **doSendBlePacket** – This *SendData* Action object will send the most recent packet from its associated Sensor Stream object (bleSensorTag) to a user specified remote packet capture server.
- **doSendEmail** – This SendEmail Action object will send an customized email to one or more recipients when the *isLightTooLow* evaluates to *true*.
- **doStartReEnableTimer** – This *StartOneShotTimer* Action object will start a 15 minute timer to reenable the sending of alarm emails.

Event objects:

- **00-onNewAntPacket** – This *NewSensorPacket* Event object will be queued every time a new packet is received from its associated Sensor Stream objects (antTempe).
- **01-onNewBlePacket** – This *NewSensorPacket* Event object will be queued every time a new packet is received from its associated Sensor Stream objects (bleSensorTag).

- **05-onReEnableTimerExpired** – This TimerExpired Event object will be queued every time its associated Timer object expires.

Rule objects:

- **00-doSendAntPacket** – This Rule object will be evaluated every time its associated Event object (00-onNewAntPacket) is queued.
- **00-doSendBlePacket** – This Rule object will be evaluated every time its associated Event object (01-onNewBlePacket) is queued.
- **01-doCheckBadLight** – This Rule object will be evaluated every time its associated Event object (01-onNewBlePacket) is queued. It has conditional logic indicating its Action objects should only be executed when the sensor data’s light level value goes below a defined threshold.
- **00-doReEnableLightCheck** – This Rule object will be evaluated every time its associated Event object (00-onReEnableTimerExpired) is queued. Its single Action object will reenable the 01-doCheckBadLight Rule object.

Full Rule object definition(s):

Figure 107 shows the full definition of the *00-doSendAntPacket* Rule object. It will be evaluated every time its triggering Event object, *00-onNewAntPacket*, gets read from the rule engine's event queue. It has no conditional State Check logic to evaluate so its Action object(s) will always be executed. When executed its single Action object, *doSendAntPacket,* will send the most recently received packet from the ANT Tempe sensor to a user specified remote packet capture server.

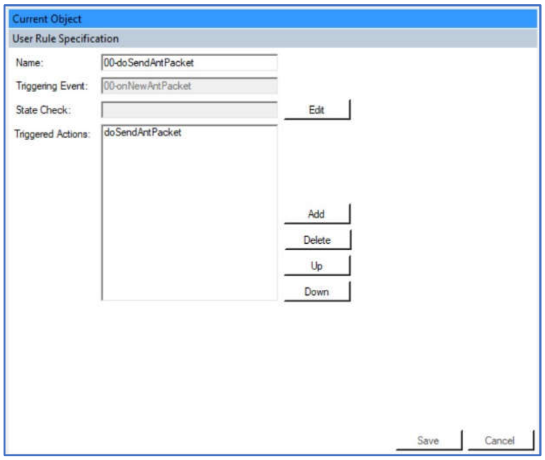

Current Object
User Rule Specification
Name: 00-doSendAntPacket
Triggering Event: 00-onNewAntPacket
State Check:
Triggered Actions: doSendAntPacket

Figure 39: The 00-doSendAntPacket Rule object definition

Figure 108 shows the full definition of the *00-doSendBlePacket* Rule object. It will be evaluated every time its triggering Event object, *01-onNewBlePacket*, gets read from the rule engine's event queue. It has no conditional State Check logic to evaluate so its Action object(s) will always be executed. When executed its single Action object, *doSendBlePacket,* will send the most recently received packet from the BLE SensorTag sensor to a user specified remote packet capture server.

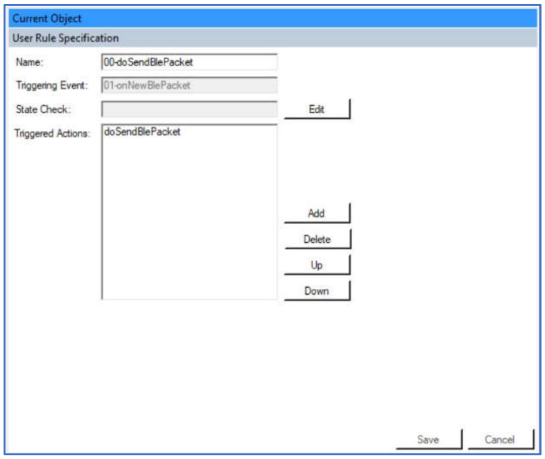

Current Object
User Rule Specification
Name: 00-doSendBlePacket
Triggering Event: 01-onNewBlePacket
State Check:
Triggered Actions: doSendBlePacket

Figure 40: The 00-doSendBlePacket Rule object definition

Figure 109 shows the full definition of the *00-doCheckBadLight* Rule object. It will be evaluated every time its triggering Event object, *01-onNewBlePacket*, gets read from the rule engine's event queue. It will be evaluated after the 00-doSendBlePacket Rule object has fully completed. It has a single conditional State object, *isLightTooLow,* which when true will cause its three Action objects to be executed. These objects will (1) send an alarm email, (2) temporarily disable the *00-doCheckBadLight* Rule object itself, and (3) set a 20 minute timer to re-enable the *00-doCheckBadLight* Rule object.

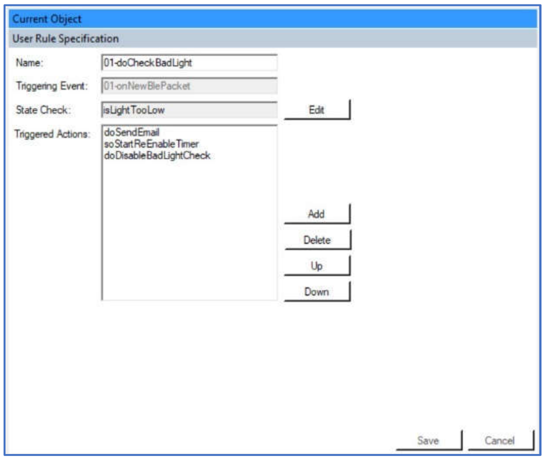

Current Object
User Rule Specification
Name: 01-doCheckBadLight
Triggering Event: 01-onNewBlePacket
State Check: isLightTooLow
Triggered Actions:
doSendEmail
soStartReEnableTimer
doDisableBadLightCheck

Figure 41: The 01-doCheckBadLight Rule object definition

Figure 110 shows the full definition of the *00-doReEnableLightCheck* Rule object. It will be evaluated every time its triggering Event object, *00-onReEnableTimerExpired*, gets read from the rule engine's event queue. This Rule object has no conditionals so its single Action object will always be executed. Its Action object will re-enable the *00-doCheckBadLight* for evaluation and execution.

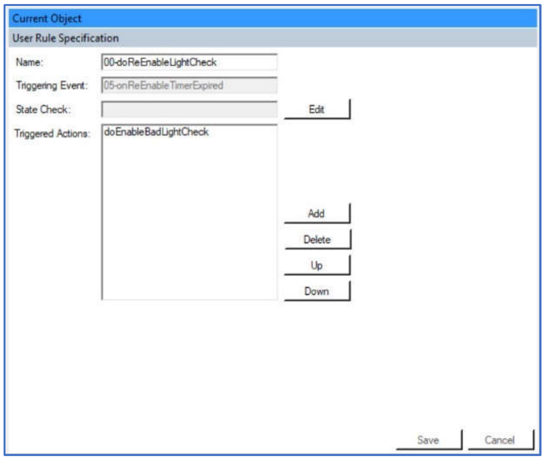
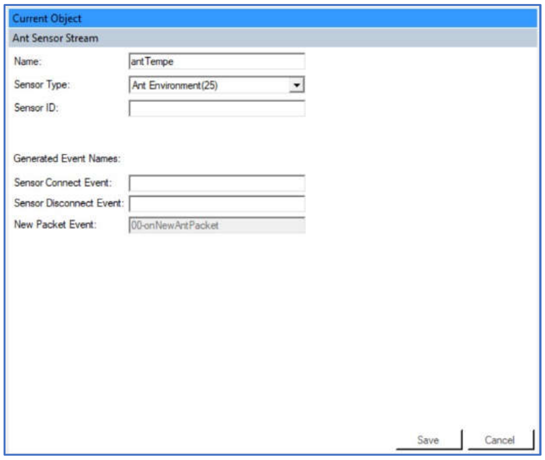

Current Object
User Rule Specification
Name: 00-doReEnableLightCheck
Triggering Event: 05-onReEnableTimerExpired
State Check:
Triggered Actions:
doEnableBadLightCheck

Figure 42: The 01-doReEnableLightCheck Rule object definition

<u>Full Sensor Stream object definition(s):</u>

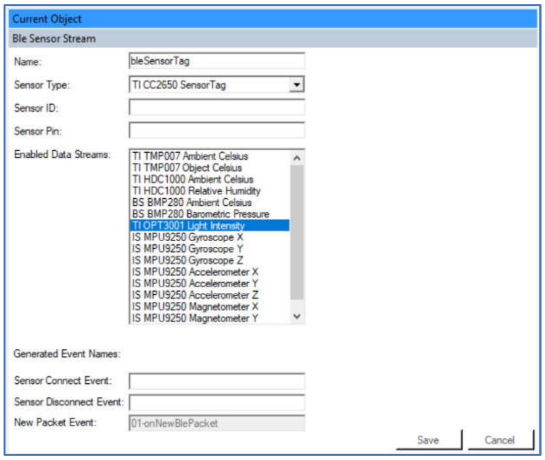
Figure 111 shows the full definition for the *antTempe* ANT Sensor Stream object. From the dropdown of supported ANT sensors a *Sensor Type of* "Ant Environment" was selected (the ANT profile for the Garmin Tempe). No specific *Sensor ID* was defined so that the SoftHub will connect to the first Tempe sensor available. Lastly, when originally creating the antTempe object a *New Packet Event* Event object was specified, *00-OnNewAntPacket*. This instructs the SoftHub to queue the named event every time a new packet is received from the Tempe sensor that connects to the SoftHub via this Sensor Stream definition.



Name: antTempe
Sensor Type: Ant Environment(25)
Sensor ID:
Generated Event Names:
Sensor Connect Event:
Sensor Disconnect Event:
New Packet Event: 00-onNewAntPacket

Figure 43: The antTempe Sensor Stream object definition

Figure 112 shows the full definition for the *bleSensorTag* BLE Sensor Stream object. From the dropdown of supported BLE sensors a *Sensor Type of* "TI CC2650 SendorTag" was selected. No specific *Sensor ID* was defined so that the SoftHub will connect to the first SensorTag sensor available. Unlike ANT sensors which will always send all sensor data available, BLE sensors must generally have supported sensor data values enabled individually. For this example, only the data from the Light Intensity sensor has been enabled from the list of values the sensor supports. Note that this is a multi-select list-box and one or more values may be selected as required.



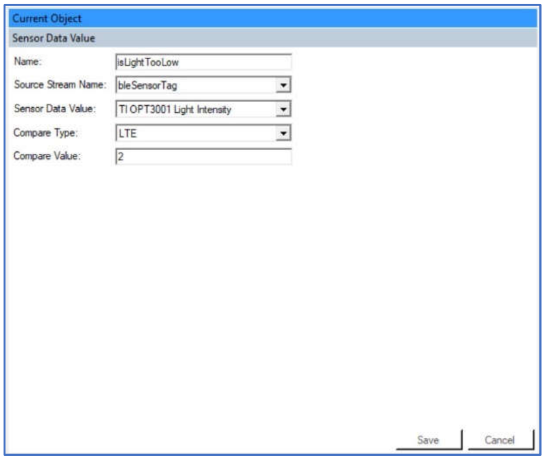
Name: bleSensorTag
Sensor Type: TI CC2650 SensorTag
Sensor ID:
Sensor Pin:
Enabled Data Streams:
TI TMP007 Ambient Celsius
TI TMP007 Object Celsius
TI HDC1000 Ambient Celsius
TI HDC1000 Relative Humidity
BS BMP280 Ambient Celsius
BS BMP280 Barometric Pressure
TI OPT3001 Light Intensity
IS MPU9250 Gyroscope X
IS MPU9250 Gyroscope Y
IS MPU9250 Gyroscope Z
IS MPU9250 Accelerometer X
IS MPU9250 Accelerometer Y
IS MPU9250 Accelerometer Z
IS MPU9250 Magnetometer X
IS MPU9250 Magnetometer Y
Generated Event Names:
Sensor Connect Event:
Sensor Disconnect Event:
New Packet Event: 01-onNewBlePacket

Figure 44: The bleSensorTag Sensor Stream object definition

Lastly, when originally creating the bleSensorTag object a *New Packet Event* Event object was specified, *01-OnNewBlePacket*. This instructs the SoftHub to queue the named event every time a new packet is received from the SensorTag sensor that connects to the SoftHub via this Sensor Stream definition.

<u>Full State object definition(s):</u>

Figure 113 shows the full definition for the *isLightTooLow* SensorDataValue State object. When executed this object will test the last received packet value specified by the *Source Stream Name* and *Sensor Data Value* per the user threshold criteria. In this example the latest received value from the SensorTag's light sensor will be checked as being less than or equal to (LTE) a threshold value of 2 (Lumens). If it is less than or equal to 2 the lights will be considered "off" amd the state object will evaluate to *true*. Otherwise it will evaluate to *false*.

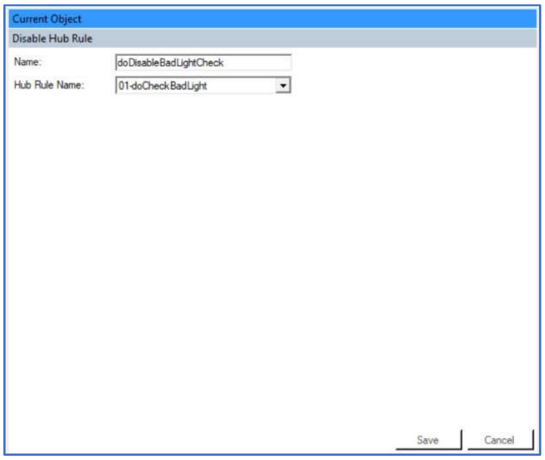


Name: isLightTooLow
Source Stream Name: bleSensorTag
Sensor Data Value: TI OPT3001 Light Intensity
Compare Type: LTE
Compare Value: 2

Figure 4513: The isLightTooLow State object definition

<u>Full Action object definition(s):</u>

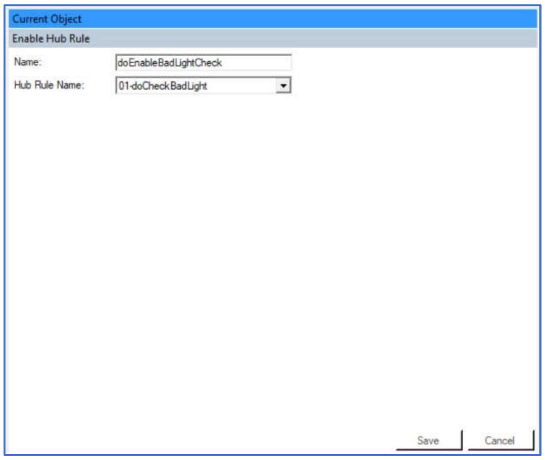
Figure 114 shows the full definition for the *doDisableBadLightCheck* DisableHubRule Action object. When executed this Action object will mark the Rule object specified in the *Hub Rule Name* dropdown as disabled. This means that when the specified Rule object's triggering event occurs, the specified Rule object will be skipped in the evaluation process. Any other rules associated with the triggering event will not be affected. A Rule object that has been disabled by a DisableHubRule Action object will remain disabled until a corresponding EnableHubRule Action object is executed specifying the same *Hub Rule Name* value.



Name: doDisableBadLightCheck
Hub Rule Name: 01-doCheckBadLight

Figure 464: The doDisableBadLightCheck Action object definition

Figure 115 shows the full definition for the *doEnableBadLightCheck* EnableHubRule Action object. When executed the EnableHubRule object will perform the inverse of the DisableHubRule object, re-enabling the Rule object chosen in the *Hub Rule Name* dropdown. It should be noted that by default all Rule objects are initially enabled so the EnableHubRule Action object should only be executed after a previous DisableHubRule Action object has been executed for the same named Rule object.

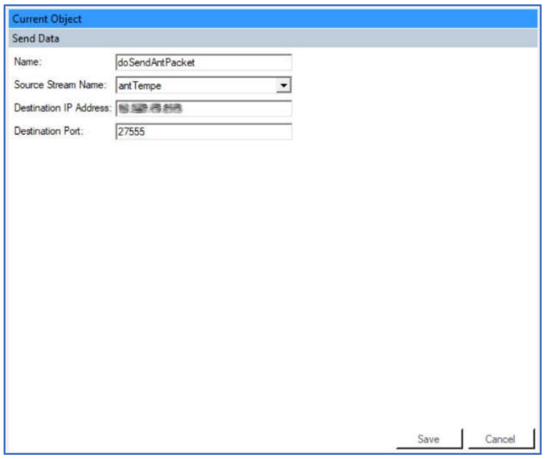


Name: doEnableBadLightCheck
Hub Rule Name: 01-doCheckBadLight

Figure 47: The doEnableBadLightCheck Action object definition

Figure 106 shows the full definition for the *doSendAntPacket* SendData Action object. When executed this object will send the latest raw packet data from the chosen Sensor Stream to the user specified IP and port address over a TLS encrypted connection.

When defining a SendData object, the *Source Stream Name* dropdown will provide a list of all Sensor Streams the user has previously defined. For this object the *antTempe* Sensor Stream object was selected. The *Destination IP Address* identifies the IP address of a remote packet capture server to forward the packets to (typically a server running the OmnIoT Packet Capture Service). The *Destination Port* identifies what port the packet capture service has been configured to listen on.

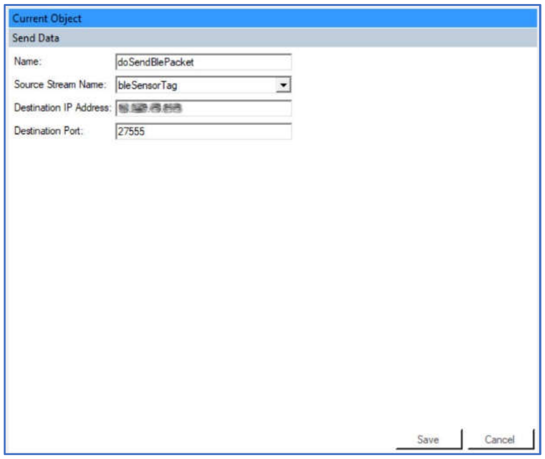


Name: doSendAntPacket
Source Stream Name: antTempe
Destination IP Address: [redacted]
Destination Port: 27555

Figure 48: The doSendAntPacket Action object definition

Figure 117 shows the full definition for the *doSendBlePacket* SendData Action object. When executed this object will send the latest raw packet data from the chosen Sensor Stream to the user specified IP and port address over a TLS encrypted connection.

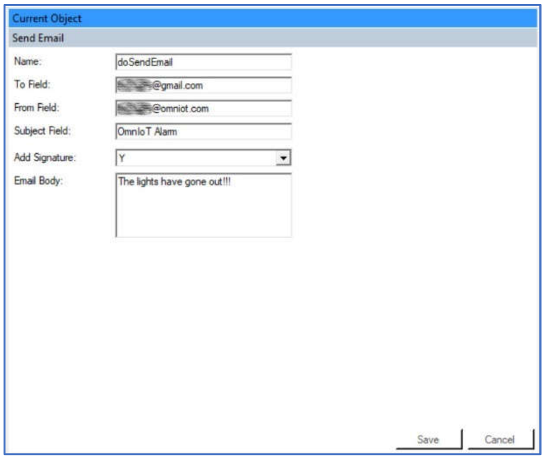
When defining a SendData object, the *Source Stream Name* dropdown will provide a list of all Sensor Streams the user has previously defined. For this object the *bleSensorTag* Sensor Stream object was selected. The *Destination IP Address* identifies the IP address of a remote packet capture server to forward the packets to (typically a server running the OmnIoT Packet Capture Service). The *Destination Port* identifies what port the packet capture service has been configured to listen on.



Name: doSendBlePacket
Source Stream Name: bleSensorTag
Destination IP Address: [redacted]
Destination Port: 27555

Figure 49: The doSendBlePacket Action object definition

Figure 118 shows the full definition for the *doSendEmail* SendData Action object. When executed the SendEmail Action object will queue an email to be sent to one or more recipients. The *To Field* should contain one or more recipients. The *From Field* should be a valid sending ID for the SMTP account specified in the Email Systems Options (detailed in the next section). The *Subject Field* and *Email Body* should contain the subject and email body to be sent respectively. Choosing “Y” in the *Add Signature* dropdown will suffix the email body with a “Time Generated” timestamp as well as the originating *Hub Device ID* specified in the Hub System Options.

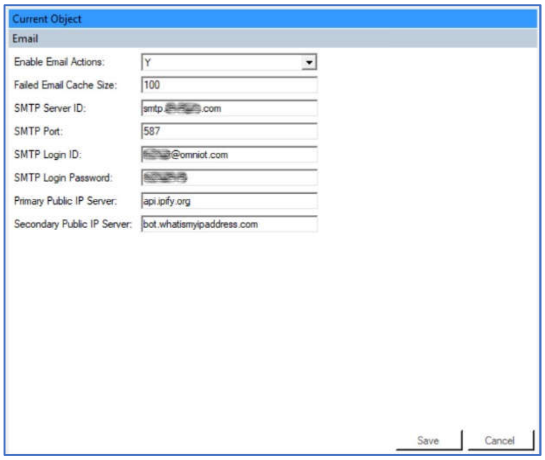


Name: doSendEmail
To Field: [redacted]@gmail.com
From Field: [redacted]@omniot.com
Subject Field: OmnIoT Alarm
Add Signature: Y
Email Body: The lights have gone out!!!

Figure 50: The doSendEmail Action object definition

Note that the SendEmail object can be used to send SMS messages as well as emails. To do so, substitute the email address with the recipient’s 10-digit phone number followed by the appropriate SMS gateway domain (e.g. 5554342345@txt.att.net).

Before creating any SendEmail Action objects, the global *Email System Options* must set to enable email actions and a valid SMTP server and account information must be provided. Figure 119 shows the fields that need to be provided. All fields are required. Typically your email service provider will supply you with the SMTP server id and port information. Additionally, valid email ID and password credentials must be provided. Email can *only* be sent via TLS secure connections, typically over port 587. Trying to send email over a non-secure connection will is not supported.



Current Object

Email

Enable Email Actions: Y

Failed Email Cache Size: 100

SMTP Server ID: smtp.[redacted].com

SMTP Port: 587

SMTP Login ID: [redacted]@omniot.com

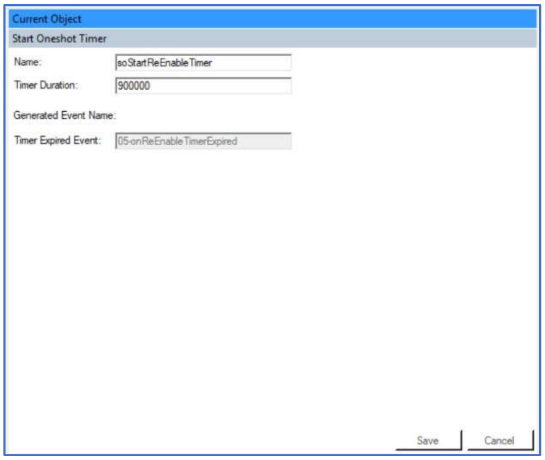
SMTP Login Password: [redacted]

Primary Public IP Server: api.ipify.org

Secondary Public IP Server: bot.whatismyipaddress.com

Figure 51: The Email System Options settings screen

Figure 120 shows the full definition for the *doStartReEnableTimer* StartOneShotTimer Action object. When executed this object will create and start a new oneshot timer object. Oneshot timer objects will run only one time and on expiration will no longer exit. For this example the *Timer Duration* has been set to 900000 millseconds (or 15 minutes). The *Timer Expired Event* dialog box defines an optional TimerExpired object to be queued each time the timer expires. Here, 15 minutes after the timer is started an *05-onReEnableTimerExpired* TimerExpired object will be be written to the rule engine's event queue.



Current Object

Start Oneshot Timer

Name: soStartReEnableTimer

Timer Duration: 900000

Generated Event Name:

Timer Expired Event: 05-onReEnableTimerExpired

Figure 52: The doStartReEnableTimer Action object definition

Example Summary:

Example 3 has added new objects to extend the functionality originally presented in Example 1 and in particular illustrates alarm condition processing and the SendEmail Action object. SoftHub rulesets can be thought of as a series of *when/if/do* statements (i.e. **WHEN** *some event occurs*, **IF** *some conditions are met*, **DO** *some action(s)*). The table below provides a summary of how the SoftHub's rule engine will process the ruleset described in Example 3:

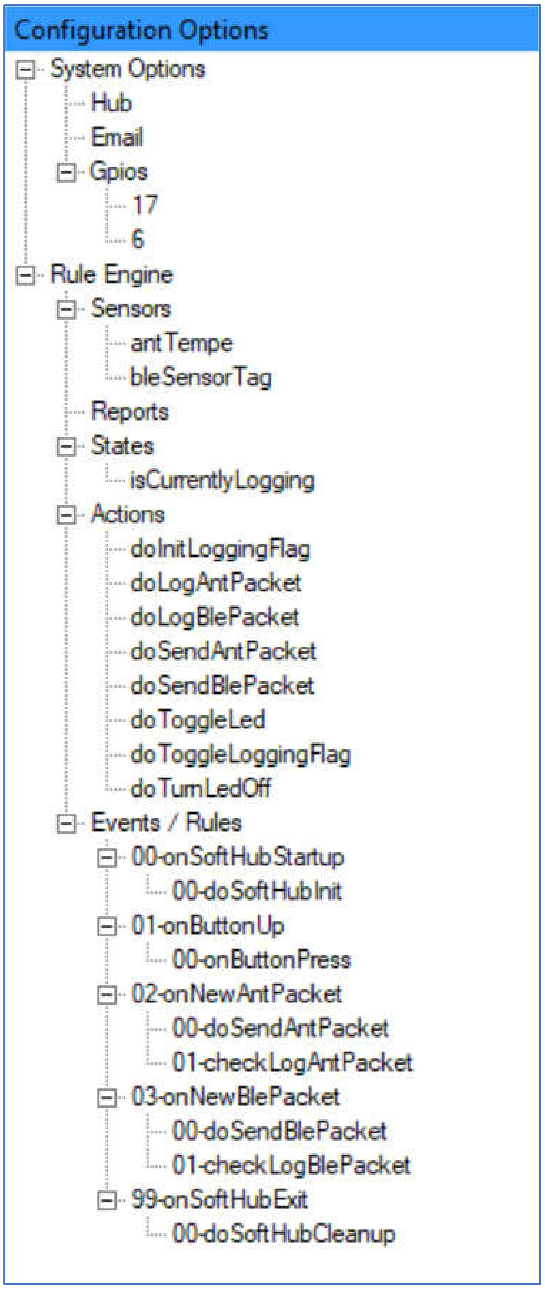
Event (When)	**Rule**	**State (If)**	**Action (Do)**
00-onNewAntPacket: A new packet has been received from the connected Tempe sensor.	00-doSendAntPacket	N/A, Do always	doSendAntPacket: Send the most recent Tempe packet to a remote packet capture server.
01-onNewBlePacket: A new packet has been received from the connected SensorTag sensor.	00-doSendBlePacket	N/A, Do always	doSendBlePacket: Send the most recent SensorTag packet to a remote packet capture server.
As Above	01-doCheckBadLight	Do only if the State object *isLightTooLow* returns *true.*	doSendEmail: Send a custom alert email to one or more recipients.
As Above	As Above	As Above	doStartReEnableTimer: Start a timer to indicate email alerts should be re-enabled.
As Above	As Above	As Above	doDisableBadLightCheck: Temporarily disable the Rule object responsible for checking light below threshold value.
05-onReEnableTimerExpired: The timer to re-enable low light threshold checking has expired.	00-doReEnableLightCheck	N/A, Do always	doEnableBadLightCheck: Re-enable the previously disabled Rule object doing the low light threshold checking.

5.4 Logging, Flags, and GPIO's

Example 4 also uses example 1 as a base but has added functionality to illustrate the use of Data Logging, Flag objects, and the manipulation of GPIO inputs and outputs. As in the previous example, new functionality not included in the original Example 1 scenario are highlighted below in *italics*. Example 4's configuration file will do the following:

- On application startup it will implicitly connect to two sensors (via the *Auto Start All Sensors* System Option), (1) an ANT Garmin Tempe sensor, and (2) a BLE TI SensorTag sensor.
- Once connected, every packet received from either sensor will be forwarded to a remote packet capture server.
- *Button input will be enabled via an input GPIO pin.*
- *A button press will toggle the logging of incoming packets to the SoftHub's internal storage.*
- *When logging is active, the SoftHub will continue to forward packets to the remote packet capture server but will also log them internally.*
- *An LED will be attached to an output GPIO and will be illuminated to indicate to the user when logging is active*.

Figure 121 illustrates the object tree view of the ruleset used to implement this scenario. Briefly, the objects required and the purpose of each are also summarized. The following sections will then describe each object and its settings in further detail.



Configuration Options

- System Options
  - Hub
  - Email
  - Gpios
    - 17
    - 6
- Rule Engine
  - Sensors
    - antTempe
    - bleSensorTag
  - Reports
  - States
    - isCurrentlyLogging
  - Actions
    - doInitLoggingFlag
    - doLogAntPacket
    - doLogBlePacket
    - doSendAntPacket
    - doSendBlePacket
    - doToggleLed
    - doToggleLoggingFlag
    - doTurnLedOff
  - Events / Rules
    - 00-onSoftHubStartup
      - 00-doSoftHubInit
    - 01-onButtonUp
      - 00-onButtonPress
    - 02-onNewAntPacket
      - 00-doSendAntPacket
      - 01-checkLogAntPacket
    - 03-onNewBlePacket
      - 00-doSendBlePacket
      - 01-checkLogBlePacket
    - 99-onSoftHubExit
      - 00-doSoftHubCleanup

Figure 53: Example 4 configuration object tree

Sensor Stream object(s):

- **antTempe** – Identifies the ANT sensor to connect to.
- **bleSensorTag** – Identifies the BLE sensor to connect to.

State objects:

- **isCurrentlyLogging** – This *FlagValue* State object will be *true* whenever the SoftHub is actively logging packet data.

Action objects:

- **doInitLoggingFlag** – This *InitializeFlag* Action object will create and initialize the Flag object used to keep track of whether logging is either enabled or disabled.
- **doLogAntPacket** – This *LogData* Action object will log the last received Ant packet from its selected Sensor Stream object (antTemp) to the SoftHub's internal storage.
- **doLogBlePacket** – This *LogData* Action object will log the last received from its selected Sensor Stream object (bleSensorTag) to the SoftHub's internal storage.
- **doSendAntPacket** – This *SendData* Action object will send the most recent packet from its associated Sensor Stream object (antTemp) to a user specified remote packet capture server.
- **doSendBlePacket** – This *SendData* Action object will send the most recent packet from its associated Sensor Stream object (bleSensorTag) to a user specified remote packet capture server.
- **doToggleLed** – This *ToggleGpioPin* Action object will toggle the voltage being sent to its associated GPIO output pin (pin 17, connected to an LED).
- **doToggleLoggingFlag** – This *ToggleFlag* Action object will toggle the logical value of its associated Flag object (the “actively logging” flag).

- **doTurnLedOff** – This *SetGpioPinLow* Action object will set the voltage to its associated GPIO output pin (pin 17, connected to a LED) to *low*.

Event objects:

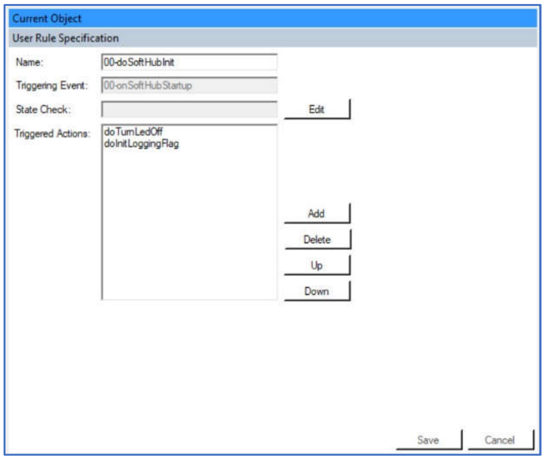
- **00-onSoftHubStartup** – This *AppStarting* Event object will be queued before any other events as the SoftHub initially start up.
- **01-onButtonUp** – This *GpioPinHigh* Event object will be queued every time the input button (GPIO Pin 6) is released after being depressed.
- **02-onNewAntPacket** – This *NewSensorPacket* Event object will be queued every time a new packet is received from its associated Sensor Stream object (antTempe).
- **03-onNewBlePacket** – This *NewSensorPacket* Event object will be queued every time a new packet is received from its associated Sensor Stream object (bleSensorTag).
- **99-onSoftHubExit** – This AppStopping Event object will be the final event queued when the SoftHub application has been asked to exit (typically when the SoftHub daemon is stopped).

Rule objects:

- **00-doSoftHubInit** – This Rule object will be evaluated every time its associated Event object (00-onSoftHubStartup) is queued.
- **00-onButtonPress** – This Rule object will be evaluated every time its associated Event object (01-onButtonUp) is queued.
- **00-doSendAntPacket** – This Rule object will be evaluated every time its associated Event object (02-onNewAntPacket) is queued.
- **01-checkLogAntPacket** – This Rule object will be evaluated every time its associated Event object (02-onNewAntPacket) is queued.
- **00-doSendBlePacket** – This Rule object will be evaluated every time its associated Event object (03-onNewBlePacket) is queued.
- **01-checkLogBlePacket** – This Rule object will be evaluated every time its associated Event object (03-onNewBlePacket) is queued.
- **01-doSoftHubCleanup** – This Rule object will be evaluated every time its associated Event object (99-onSoftHubExit) is queued.

Full Rule object definition(s):

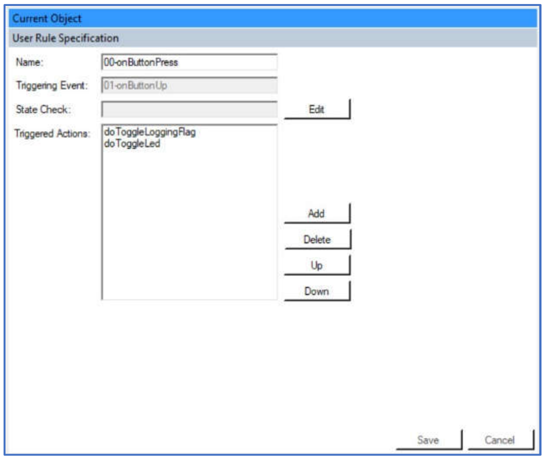
Figure 122 shows the full definition of the *00-doSoftHubInit* Rule object. It will be evaluated when its triggering Event object, *00-onSoftHubStartup*, gets read from the rule engine's event queue. It has no State Check logic to evaluate so its Action object(s) will always be executed. When executed its two Action objects will (1) turn *off* the attached LED and, (2) create and initialize a Flag object that will keep track of when the SoftHub is actively logging incoming packets (the initial state is *false*, not logging).



Name: 00-doSoftHubInit
Triggering Event: 00-onSoftHubStartup
State Check:
Triggered Actions: doTurnLedOff
doInitLoggingFlag

Figure 54: The 00-doSoftHubInit Rule object definition

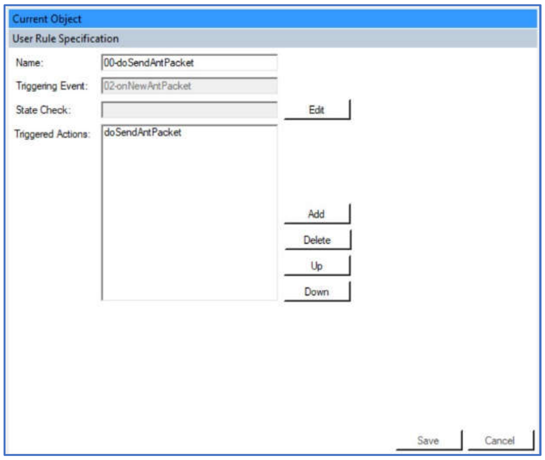
Figure 123 shows the full definition of the *00-onButtonPress* Rule object. It will be evaluated every time its triggering Event object, *01-onButtonUp*, gets read from the rule engine's event queue. It has no State Check logic to evaluate so its Action object(s) will always be executed. Its two Action objects will (1) toggle the "actively logging" Flag and, (2) toggle the attached LED's illumination state.



Name: 00-onButtonPress
Triggering Event: 01-onButtonUp
State Check:
Triggered Actions: doToggleLoggingFlag
doToggleLed

Figure 55: The 00-onButtonPress Rule object definition

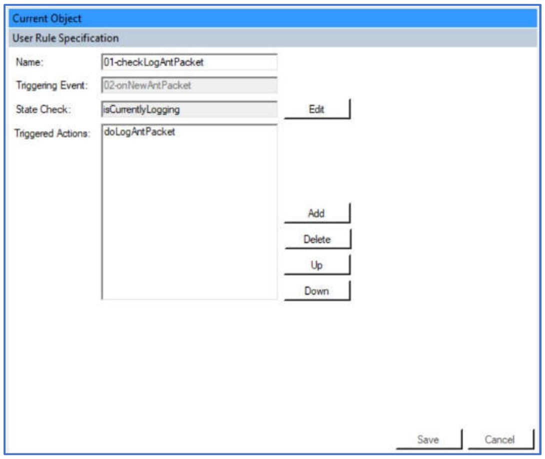
Figure 124 shows the full definition of the *00-doSendAntPacket* Rule object. It will be evaluated every time its triggering Event object, *02-onNewAntPacket*, gets read from the rule engine's event queue. It has no State Check logic to evaluate so its Action object(s) will always be executed. When executed its single Action object, *doSendAntPacket,* will send the most recently received packet from the ANT Tempe sensor to a user specified remote packet capture server.



Name: 00-doSendAntPacket
Triggering Event: 02-onNewAntPacket
State Check:
Triggered Actions: doSendAntPacket

Figure 56: The 00-doSendAntPacket object definition

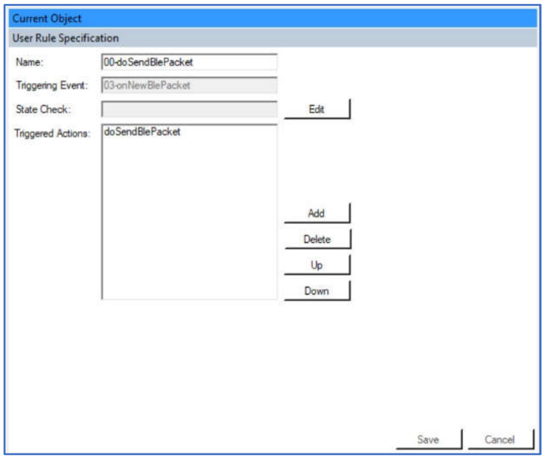
Figure 125 shows the full definition of the *01-doCheckLogAntPacket* Rule object. This rule will be evaluated every time its associated Event object, *02-onNewAntPacket*, gets read from the rule engine's event queue. It has a single stand-alone FlagState State object, *isCurrentlyLogging*, controlling its execution. When evaluated, if its State object returns a value of *true*, the action object *doLogAntPacket* will be executed and the most recently received Tempe packet will be logged to the SoftHub's internal storage.



Name: 01-checkLogAntPacket
Triggering Event: 02-onNewAntPacket
State Check: isCurrentlyLogging
Triggered Actions: doLogAntPacket

Figure 57: The 00-doCheckLogAntPacket Rule object definition

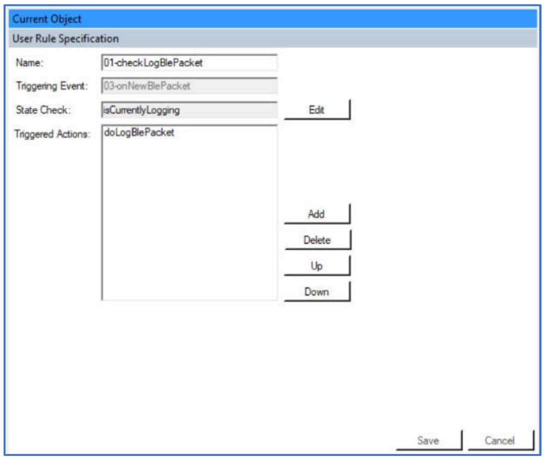
Figure 126 shows the full definition of the *00-doSendBlePacket* Rule object. It will be evaluated every time its triggering Event object, *03-onNewBlePacket*, gets read from the rule engine's event queue. It has no State Check logic to evaluate so the rule's Action object(s) will always be executed. When executed its single Action object, *doSendBlePacket,* will send the most recently received packet from the BLE SensorTag sensor to a user specified remote packet capture server.



Name: 00-doSendBlePacket
Triggering Event: 03-onNewBlePacket
State Check:
Triggered Actions: doSendBlePacket

Figure 58: The 00-doSendBlePacket object definition

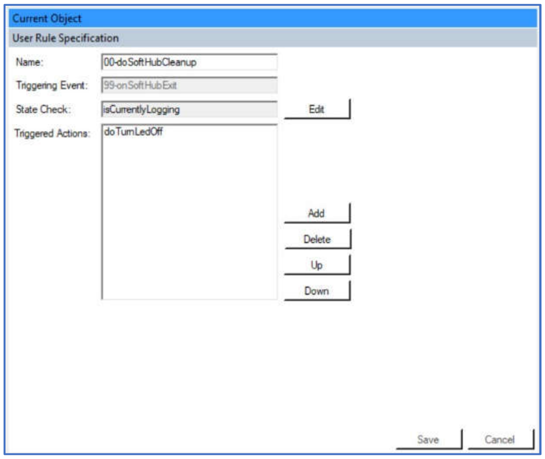
Figure 127 shows the full definition of the *01-doCheckLogBlePacket* Rule object. This rule will be evaluated every time its associated Event object, *03-onNewBlePacket*, gets queued. It has a single stand-alone FlagState State object, *isCurrentlyLogging*, controlling its execution. When evaluated, if its State object returns a value of *true*, the action object *doLogBlePacket* will be executed and the most recently received SersorTag packet will be logged to the SoftHub's internal storage.



Name: 01-checkLogBlePacket
Triggering Event: 03-onNewBlePacket
State Check: isCurrentlyLogging
Triggered Actions: doLogBlePacket

Figure 59: The 00-doCheckLogBlePacket Rule object definition

Figure 128 shows the full definition of the *00-doSoftHubCleanup* Rule object. This rule will be evaluated every time its associated Event object, *99-onSoftHubExit*, gets queued. It has a single stand-alone FlagState State object, *isCurrentlyLogging*, controlling its execution. When evaluated, if its State object returns a value of *true*, the action object doTurnLedOff will turn the "actively logging" LED off before the SoftHub application exits.

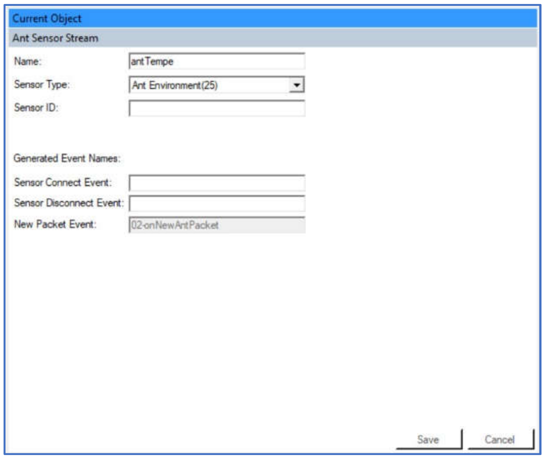


Name: 00-doSoftHubCleanup
Triggering Event: 99-onSoftHubExit
State Check: isCurrentlyLogging
Triggered Actions: doTurnLedOff

Figure 60: The 00-doSoftHubCleanup object definition

Full Sensor Stream object definition(s):

Figure 129 shows the full definition for the *antTempe* ANT Sensor Stream object. From the dropdown of supported ANT sensors a *Sensor Type of* “Ant Environment” was selected (the ANT profile for the Garmin Tempe). No specific *Sensor ID* was defined so that the SoftHub will connect to the first Tempe sensor available. Lastly, when originally creating the antTempe object a *New Packet Event* Event object was specified, *02-OnNewAntPacket*. This instructs the SoftHub to queue the named event every time a new packet is received from the Tempe sensor that connects to the SoftHub via this Sensor Stream definition.



Ant Sensor Stream

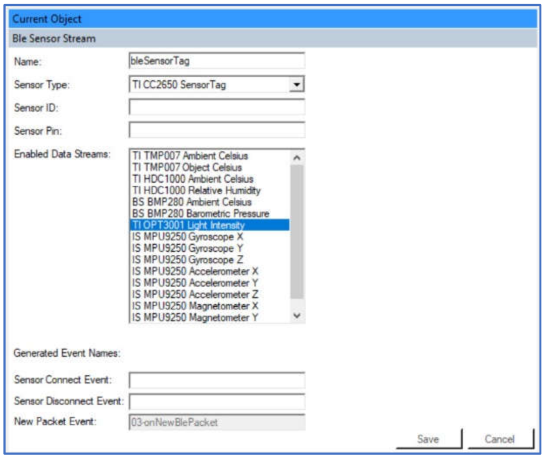
Name: antTempe
Sensor Type: Ant Environment(25)
Sensor ID:

Generated Event Names:

Sensor Connect Event:
Sensor Disconnect Event:
New Packet Event: 02-onNewAntPacket

Figure 61: The antTempe Sensor Stream object definition

Figure 130 shows the full definition for the *bleSensorTag* BLE Sensor Stream object. From the dropdown of supported BLE sensors a *Sensor Type of* “TI CC2650 SendorTag” was selected. No specific *Sensor ID* was defined so that the SoftHub will connect to the first SensorTag sensor available. Unlike ANT sensors which will always send all sensor data available, BLE sensors must generally have supported sensor data values enabled individually. For this example, only the data from the Light Intensity sensor has been enabled from the list of values the sensor supports. Note that this is a multi-select list-box and one or more values may be selected as required.



Ble Sensor Stream

Name: bleSensorTag
Sensor Type: TI CC2650 SensorTag
Sensor ID:
Sensor Pin:
Enabled Data Streams:

- TI TMP007 Ambient Celsius
- TI TMP007 Object Celsius
- TI HDC1000 Ambient Celsius
- TI HDC1000 Relative Humidity
- BS BMP280 Ambient Celsius
- BS BMP280 Barometric Pressure
- TI OPT3001 Light Intensity
- IS MPU9250 Gyroscope X
- IS MPU9250 Gyroscope Y
- IS MPU9250 Gyroscope Z
- IS MPU9250 Accelerometer X
- IS MPU9250 Accelerometer Y
- IS MPU9250 Accelerometer Z
- IS MPU9250 Magnetometer X
- IS MPU9250 Magnetometer Y

Generated Event Names:

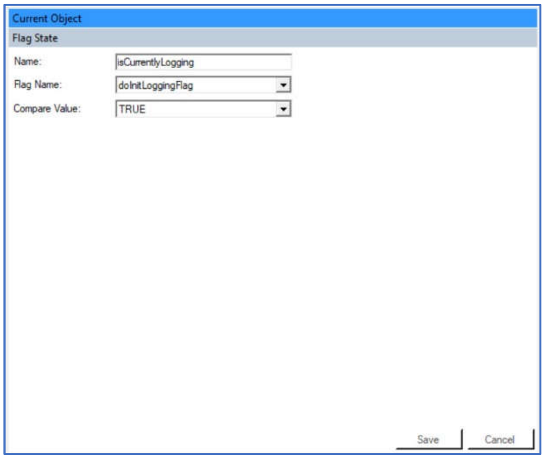
Sensor Connect Event:
Sensor Disconnect Event:
New Packet Event: 03-onNewBlePacket

Figure 62: The bleSensorTag Sensor Stream object definition

Lastly, when originally creating the bleSensorTag object a *New Packet Event* Event object was specified, *03-OnNewBlePacket*. This instructs the SoftHub to queue the named event every time a new packet is received from the SensorTag sensor that connects to the SoftHub via this Sensor Stream definition.

Full State object definition(s):

Figure 131 shows the full definition for the *isCurrentlyLogging* FlagState State object. When executed this object will test whether the value returned from the Flag object created via the *doInitLoggingFlag* Action object matches the value the user has specified in the *Compare Value* dropdown dialog box. In this example the State object will return *true* when the "actively logging" flag is *true*.

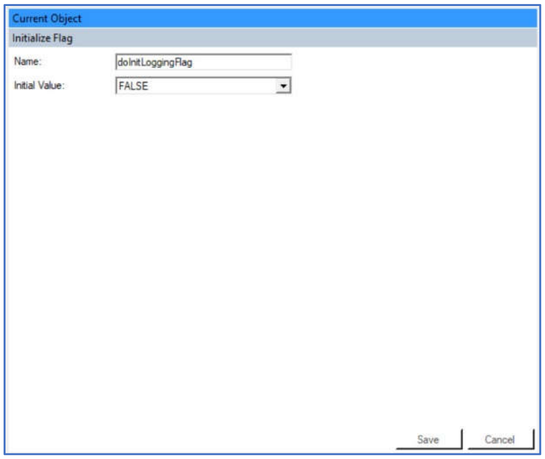


Name: isCurrentlyLogging
Flag Name: doInitLoggingFlag
Compare Value: TRUE

Figure 63: The isCurrentlyLogging State object definition

Full Action object definition(s):

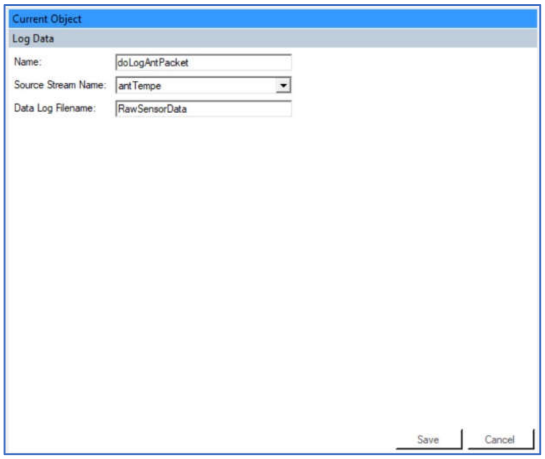
Figure 132 shows the full definition for the *doInitLoggingFlag* InitializeFlag Action object. When executed this Action will create a Flag object with the specified *Name* and set its initial value as chosen by the user in the *Initial Value* dropdown dialog box.



Name: doInitLoggingFlag
Initial Value: FALSE

Figure 64: The doInitLoggingFlag Action object definition

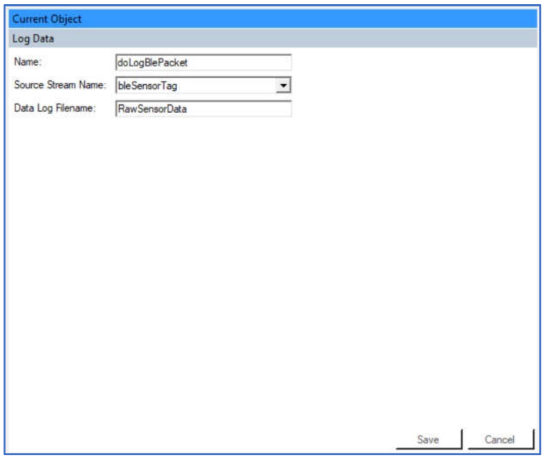
Figure 133 shows the full definition for the *doLogAntPacket* LogData Action object. When executed this Action will log the most recent packet received from the Sensor Stream chosen from the *Source Stream Name* dropdown dialog box to the user specified file named in the *Data Log Filename* text box. In this example the most recent packet received from the Tempe sensor will be written to a file named *RawSensorData.log* (the ".log" suffix is always added). Users can override the default log file directory via the *Log File Directory* text box of the Hub Systems Options dialog.



Name: doLogAntPacket
Source Stream Name: antTempe
Data Log Filename: RawSensorData

Figure 65: The doLogAntPacket Action object definition

Figure 134 shows the full definition for the *doLogBlePacket* LogData Action object. When executed this Action will log the most recent packet received from the Sensor Stream chosen from the *Source Stream Name* dropdown dialog box to the user specified file named in the *Data Log Filename* text box. In this example the most recent packet received from the SensorTag sensor will be written to a file named *RawSensorData.log* (the ".log" suffix is always added). Users can override the log file directory via the *Log File Directory* text box of the Hub Systems Options dialog.

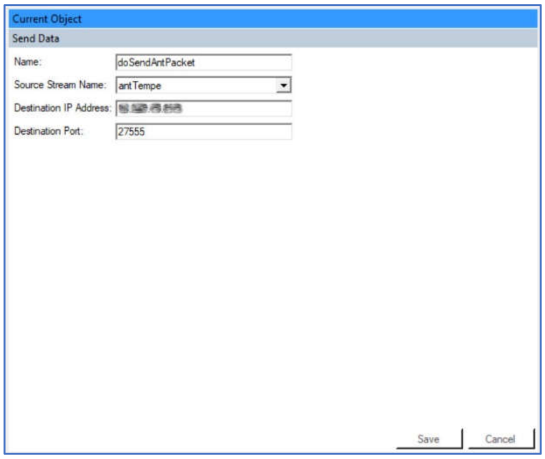


Name: doLogBlePacket
Source Stream Name: bleSensorTag
Data Log Filename: RawSensorData

Figure 66: The doLogBlePacket Action object definition

Figure 135 shows the full definition for the *doSendAntPacket* SendData Action object. When executed this object will send the latest packet data from the chosen Sensor Stream to the user specified IP and port address over a TLS encrypted connection.

When defining a SendData object, the Source Stream *Name* dropdown will provide a list of all Sensor Streams the user has previously defined. For this object the *antTempe* Sensor Stream object was selected. The *Destination IP Address* identifies the IP address of a remote packet capture server to forward the packets to (typically a server running the OmnIoT Packet Capture



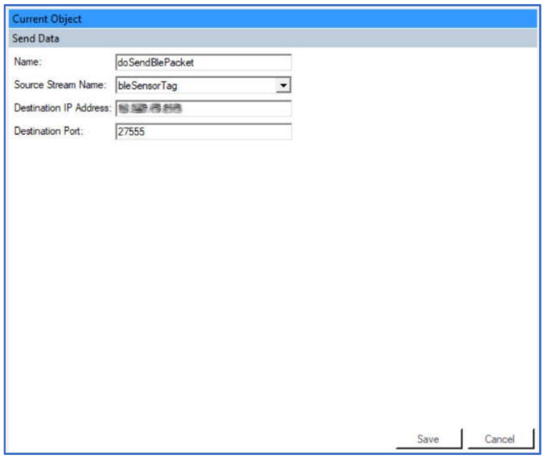
Name: doSendAntPacket
Source Stream Name: antTempe
Destination IP Address: [redacted]
Destination Port: 27555

Figure 67: The doSendAntPacket Action object definition

Service). The *Destination Port* identifies what port the packet capture service has been configured to listen on.

Figure 136 shows the full definition for the *doSendBlePacket* SendData Action object. When executed this object will send the latest packet data from the chosen Sensor Stream to the user specified IP and port address over a TLS encrypted connection.

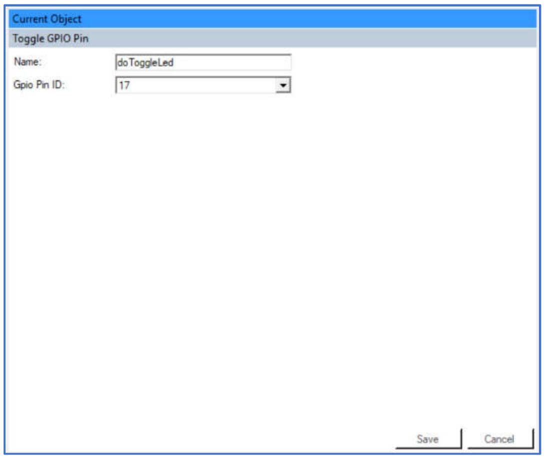
When defining a SendData object, the Source Stream *Name* dropdown will provide a list of all Sensor Streams the user has previously defined. For this object the *bleSensorTag* Sensor Stream object was selected. The *Destination IP Address* identifies the IP address of a remote packet capture server to forward the packets to (typically a server running the OmnIoT Packet Capture Service). The *Destination Port* identifies what port the packet capture service has been configured to listen on.



Name: doSendBlePacket
Source Stream Name: bleSensorTag
Destination IP Address: [redacted]
Destination Port: 27555

Figure 68: The doSendBlePacket Action object definition

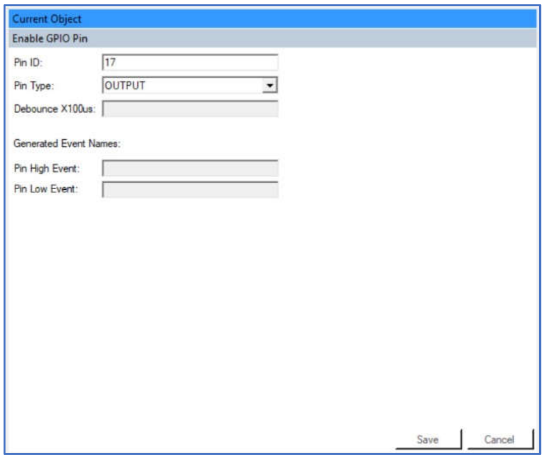
Figure 137 shows the full definition for the *doToggleLed* ToggleGpioPin Action object. When executed this Action object will toggle the voltage on the GPIO pin the user chooses from the *GPIO Pin ID* dropdown dialog box. This dialog box will enumerate all pins previously enabled as *output* pins in the GPIO System Options dialog (shown below). For example 4, pin 17 should be connected to a physical LED.



Name: doToggleLed
Gpio Pin ID: 17

Figure 69: The doToggleLed Action object definition

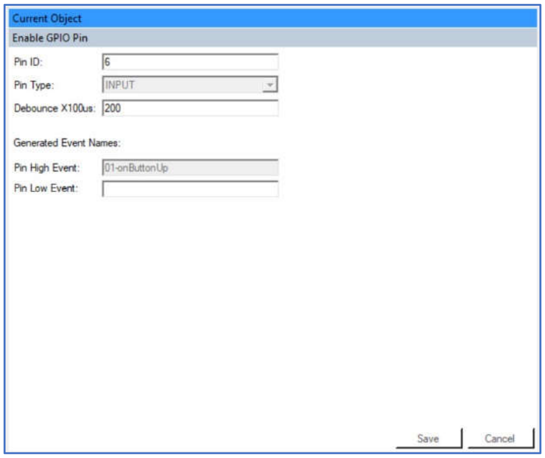
Figure 138 shows the Enable GPIO Pin System Options dialog screen used to enable pin 17 as an *output* pin. To enable a new output GPIO pin the user can right click on the *System Options > GPIOs* tree node and select *New*. To run Example 4, pin 17 should be connected to a physical LED.



Pin ID: 17
Pin Type: OUTPUT
Debounce X100us:
Generated Event Names:
Pin High Event:
Pin Low Event:

Figure 70: The Enable GPIO Pin System Option dialog

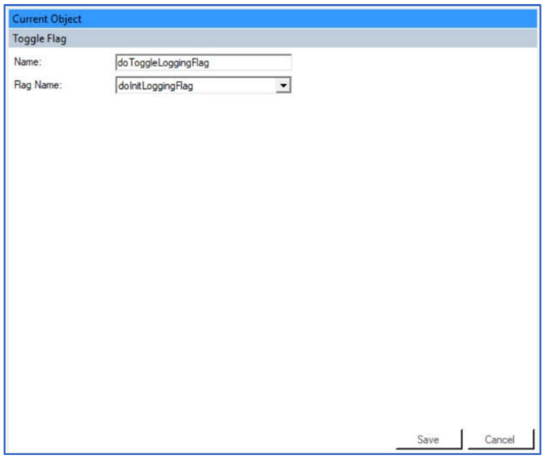
Figure 139 shows the Enable GPIO Pin System Options dialog screen used to enable pin 17 as an *output* pin. To enable a new input GPIO pin the user can right click on the *System Options > GPIOs* tree node and select *New*. To run Example 4, pin 6 should be connected to a physical button device. Input GPIO pins may optionally have *Pin High* or *Pin Low* Event objects defined to indicate voltage transition events. Setting the Pin High Event textbox to *01-onButtonUp* will cause a GpioPinHigh Event object to be queued each time the button is released after being depressed.



Pin ID: 6
Pin Type: INPUT
Debounce X100us: 200
Generated Event Names:
Pin High Event: 01-onButtonUp
Pin Low Event:

Figure 71: The Enable GPIO Pin System Option dialog

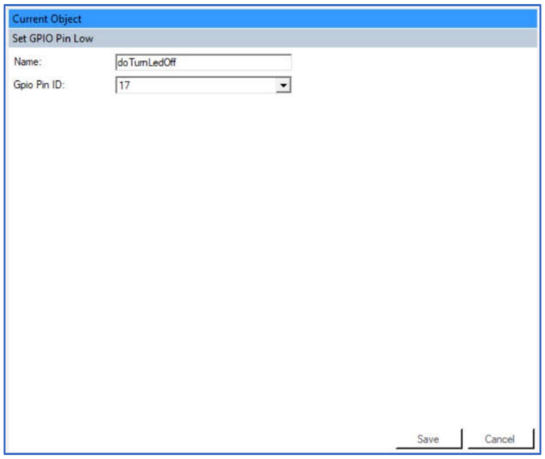
Figure 140 shows the full definition for the *doToggleLoggingFlag* ToggleFlag Action object. When executed this Action will toggle the logical value of the flag chosen in the Flag Name dropdown dialog box, which will list all Flag object previously created. In this example the Flag object created by the *doInitLoggingFlag* Action object will be toggled.



Name: doToggleLoggingFlag
Flag Name: doInitLoggingFlag

Figure 72: The doToggleLoggingFlag Action object definition

Figure 141 shows the full definition for the *doTurnLedOff* SetGpioPinLow Action object. When executed this Action will set the voltage currently being applied to an output pin to *low*. All GPIO pins previously enabled as output pins int the *GPIO System Options* dialog will be listed in the *GPIO Pin ID* dropdown box. In this example, pin 17’s voltage will be set to low (i.e. LED off).



Name: doTurnLedOff
Gpio Pin ID: 17

Figure 73: The doTurnOffLed Action object definition

Example Summary:

Example 4 has added new objects to extend the functionality presented in Example 1. Example 4 adds GPIO input and output pin manipulation, Flag objects, and sensor Data Logging objects. SoftHub rulesets can be thought of as a series of *when/if/do* statements (i.e. **WHEN** *some event occurs*, **IF** *some conditions are met*, **DO** *some action(s)*). The table below provides a summary of how the SoftHub's rule engine will process the ruleset described in Example 4:

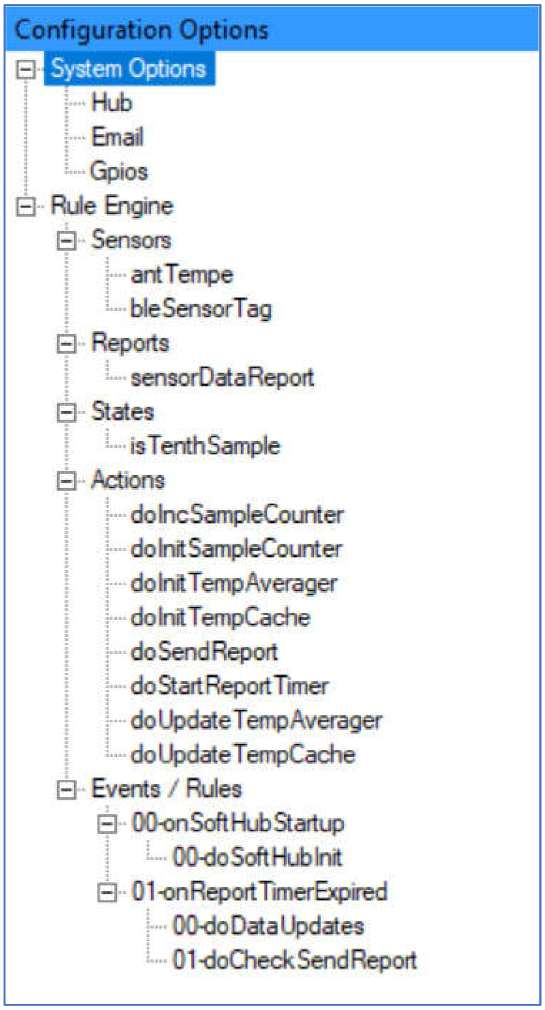
Event (When)	**Rule**	**State (If)**	**Action (Do)**
00-onSoftHubStartup: The SoftHub application has started.	00-doSoftHubInit	N/A, Do always	doTurnLedOff: Set the initial LED voltage as low (LED off).
As above	As above	As above	doInitLoggingFlag: Create and initialize a Flag object to indicate if we are currently logging.
01-onButtonUp: An input button has been depressed and released.	00-onButtonPress	N/A, Do always	doToggleLoggingFlag: Invert the logical value of the "currently logging" flag.
As above	As above	As above	doToggleLed: Invert the voltage on the GPIO pin connected to the "currently logging" LED.
02-onNewAntPacket: A new packet has been received from the connected Tempe sensor.	00-doSendAntPacket	N/A, Do always	doSendAntPacket: Send the most recently received Tempe packet to a remote packet capture server.
As above	01-checkLogAntPacket	Do only if executing State object *isCurrentlyLogging* returns *true.*	doLogAntPacket: Log the most recently received Tempe packet to the SoftHub's internal storage.
03-onNewBlePacket: A new packet has been received from the connected SensorTag sensor.	00-doSendBlePacket	N/A, Do always	doSendBlePacket: Send the most recently received SensorTag packet to a remote packet capture server.
As above	01-checkLogBlePacket	Do only if executing State object *isCurrentlyLogging* returns *true*.	doLogBlePacket: Log the most recently received SensorTag packet to the SoftHub's internal storage.
99-onSoftHubExit: The SoftHub has received a shutdown command and will exit.	00-doSoftHubCleanup	Do only if executing State object *isCurrentlyLogging* returns *true*.	doTurnLedOff: Turn off the "currently logging" LED before exiting the application.

5.5 Counters, Averagers, Cache, and Custom Report packets

Example 5 again is again a variation on example 1 and has added functionality to illustrate the use of Counter objects, Averager objects, Cache objects, and Report objects. Example 5's configuration file will do the following:

- On application startup it will implicitly connect to two sensors (via the *Auto Start All Sensors* System Option), (1) an ANT Garmin Tempe sensor, and (2) a BLE TI SensorTag sensor.
- It will define a custom User Report packet, in XML format, to include the following data (1) the latest Lumen value from the BLE SensorTag's light sensor, (2) the latest Temperature value from the ANT Tempe sensor, (3) the average temperature from the Tempe sensor over the last 5 minutes, and (4) a cache of the 10 data points we used to compute the Temperature average.
- Data points to create the temperature average will be sampled every 30 seconds.
- User Report packets (as described above) will be sent to a remote packet capture server every 5 minutes.

Figure 142 illustrates the object tree view of the ruleset used to implement this scenario. Briefly, the objects and the purpose of each are also summarized. The following sections will then describe each object and its settings in further detail.



Configuration Options

- System Options
  - Hub
  - Email
  - Gpios
- Rule Engine
  - Sensors
    - antTempe
    - bleSensorTag
  - Reports
    - sensorDataReport
  - States
    - isTenthSample
  - Actions
    - doIncSampleCounter
    - doInitSampleCounter
    - doInitTempAverager
    - doInitTempCache
    - doSendReport
    - doStartReportTimer
    - doUpdateTempAverager
    - doUpdateTempCache
  - Events / Rules
    - 00-onSoftHubStartup
      - 00-doSoftHubInit
    - 01-onReportTimerExpired
      - 00-doDataUpdates
      - 01-doCheckSendReport

Figure 74: Example 5 configuration object tree

Sensor Stream object(s):

- **antTempe** – Identifies the ANT sensor to connect to.
- **bleSensorTag** – Identifies the BLE sensor to connect to.

User Report object(s):

- **sensorDataReport** – Provides a template for a user defined custom report packet.

State object(s):

- **isTenthSample** – This *CounterValue* State object is *true* when the sample counter created by the doInitSampleCounter Action object has met its threshold (modulo) and returned to zero.

Action object(s):

- **doIncSampleCounter** – This *IncrementCounter* Action object is used to increment the sample counter created by the *doInitSampleCounter* Action object.
- **doInitSampleCounter** – This *InitializeCounter* Action object creates and initializes a counter that will return to zero (modulo) on every tenth sample.
- **doInitTempAverager** – This *InitDataAverager* Action object will create an Averager object that will maintain a 10 sample moving average of the temperature values received from the *antTempe* Sensor Stream.
- **doInitTempCache** – This *InitSensorDataCache* Action object will create a 10 sample circular Cache object to hold temperature samples received from the *antTempe* Sensor Stream.
- **doSendReport** – When executed, this *SendReport* Action object will create the custom report packet described by the *sensorDataReport* User Report object. Once created it will be transmitted to the user specified remote packet capture server.
- **doStartReportTimer** – This *StartContinuousTimer* Action object will start a timer that will continuously expire and restart every 30 seconds.

- **doUpdateTempAverager** – This *UpdateDataAverager* Action object will update the Averager object created by the *doInitTempAverager* Action object with the most recent temperature value received from the *antTempe* Sensor Stream.
- **doUpdateTempCache** – This *UpdateSensorDataCache* Action object will update the Cache object created by the *doInitTempCache* Action object with the most recent temperature value received from the *antTempe* Sensor Stream.

Event object(s):

- **00-onSoftHubStartup** – This *AppStarting* Event object will be queued as the first event to be processed when the SoftHub application initially starts.
- **01-onReportTimerExpired** – This *TimerExpired* Event object will be queued at 30 second intervals as the Timer object created by the *doStartReportTimer* Action object expires.

Rule object(s):

- **00-doSoftHubInit** – This Rule object will be evaluated every time its associated Event object (00-onSoftHubStartup) is queued.
- **00-doDataUpdates** – This Rule object will be evaluated every time its associated Event object (01-onReportTimerExpired) is queued.
- **01-doCheckSendReport** – This Rule object will be evaluated every time its associated Event object (01-onReportTimerExpired) is queued. Its execution is conditional based on the *isTenthSample* State object.

Full Rule object definition(s):

Figure 143 shows the full definition of the *00-doSoftHubInit* Rule object. It will be evaluated when its triggering Event object, *00-onSoftHubStartup*, gets read from the rule engine's event queue. It has no State Check logic to evaluate so its Action object(s) will always be executed. When executed its four Action objects will (1) create and initialize a Sensor Data Averager object, (2) create and initialize a Sensor Data Cache object, (3) create and initialize a Counter object, and (4) create and start a continuous 30 second Timer object.

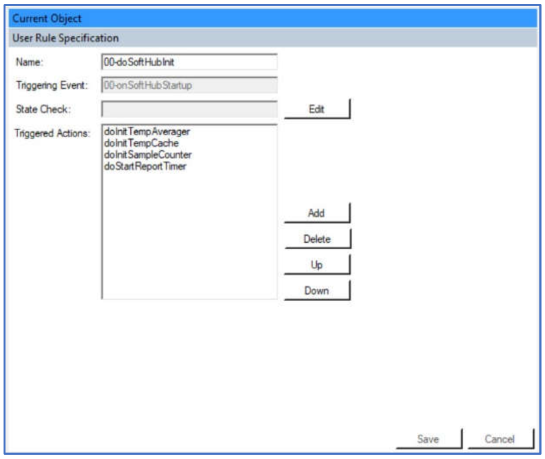

Current Object
User Rule Specification
Name: 00-doSoftHubInit
Triggering Event: 00-onSoftHubStartup
State Check:
Triggered Actions:
doInitTempAverager
doInitTempCache
doInitSampleCounter
doStartReportTimer

Figure 75: The 00-doSoftHubInit Rule object definition

Figure 144 shows the full definition of the *00-onDataUpdates* Rule object. It will be evaluated every time its triggering Event object, *01-onReportTimerExpired*, gets read from the rule engine's event queue. It has no State Check logic to evaluate so the its Action object(s) will always be executed. Its three Action objects will (1) update the Averager object with the latest Tempe Temperature value, (2) update the Cache object with the latest Tempe Temperature value, and (3) increment the sample Counter object's value by 1.

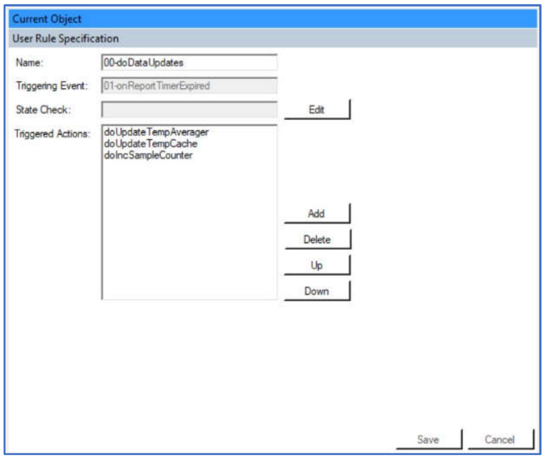
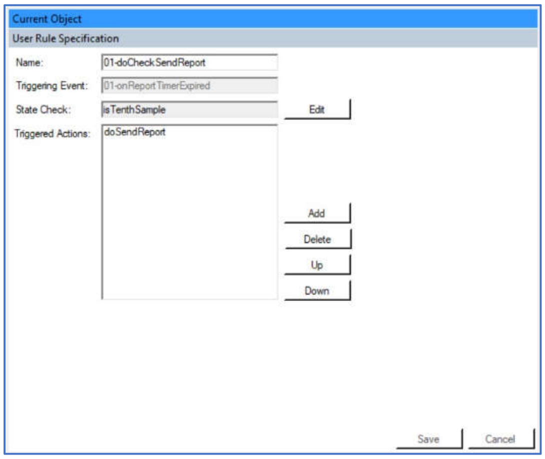

Current Object
User Rule Specification
Name: 00-doDataUpdates
Triggering Event: 01-onReportTimerExpired
State Check:
Triggered Actions:
doUpdateTempAverager
doUpdateTempCache
doIncSampleCounter

Figure 76: The 00-onButtonPress Rule object definition

Figure 145 shows the full definition of the *01-onCheckSendReport* Rule object. It will be evaluated every time its triggering Event object, *01-onReportTimerExpired* gets read from the rule engine's event queue. It has a single standalone State object to be evaluated. Only when the *isTenthSample* State object's value is *true* will its Action object be executed. In this example *isTenthSample* will be true every 10th 30 second interval (i.e. every 5 minutes). At that time the *doSendReport* SendReport Action object will create and send the *sensorDataReport* report packet to a user specified remote packet capture server.

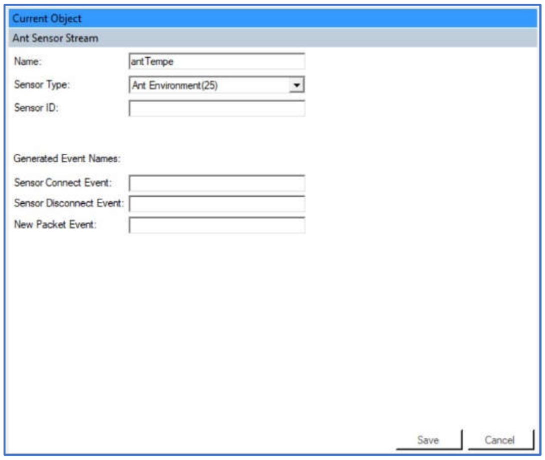


Name: 01-doCheckSendReport
Triggering Event: 01-onReportTimerExpired
State Check: isTenthSample
Triggered Actions: doSendReport

Figure 77: The 01-doCheckSendReport Rule object definition

Full Sensor Stream object definition(s):

Figure 146 shows the full definition for the *antTempe* ANT Sensor Stream object. From the dropdown of supported ANT sensors a *Sensor Type of* "Ant Environment" was selected (the ANT profile for the Garmin Tempe). No specific *Sensor ID* was defined so that the SoftHub will connect to the first Tempe sensor available.

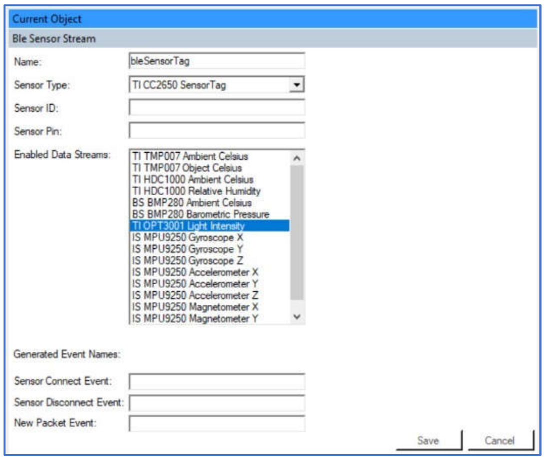


Name: antTempe
Sensor Type: Ant Environment(25)
Sensor ID:

Generated Event Names:
Sensor Connect Event:
Sensor Disconnect Event:
New Packet Event:

Figure 78: The antTempe Sensor Stream object definition

Figure 147 shows the full definition for the *bleSensorTag* BLE Sensor Stream object. From the dropdown of supported BLE sensors a *Sensor Type of* "TI CC2650 SendorTag" was selected. No specific *Sensor ID* was defined so that the SoftHub will connect to the first SensorTag sensor available. Unlike ANT sensors which will always send all sensor data available, BLE sensors must generally have supported sensor data values enabled individually. For this example, only the data from the Light Intensity sensor has been enabled from the list of values the sensor supports. Note that this is a multi-select list-box and one or more values may be selected as required.



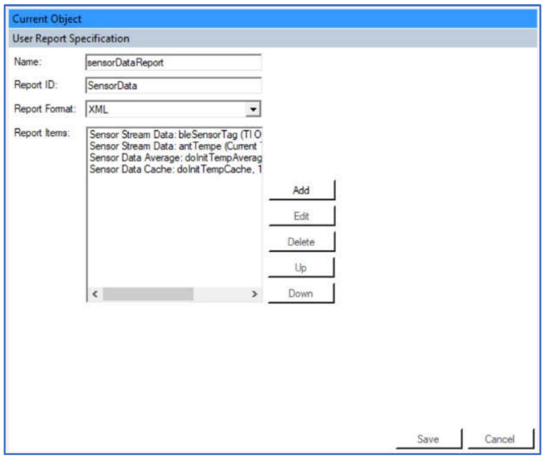
Name: bleSensorTag
Sensor Type: TI CC2650 SensorTag
Sensor ID:
Sensor Pin:
Enabled Data Streams:
- TI TMP007 Ambient Celsius
- TI TMP007 Object Celsius
- TI HDC1000 Ambient Celsius
- TI HDC1000 Relative Humidity
- BS BMP280 Ambient Celsius
- BS BMP280 Barometric Pressure
- TI OPT3001 Light Intensity
- IS MPU9250 Gyroscope X
- IS MPU9250 Gyroscope Y
- IS MPU9250 Gyroscope Z
- IS MPU9250 Accelerometer X
- IS MPU9250 Accelerometer Y
- IS MPU9250 Accelerometer Z
- IS MPU9250 Magnetometer X
- IS MPU9250 Magnetometer Y

Generated Event Names:
Sensor Connect Event:
Sensor Disconnect Event:
New Packet Event:

Figure 79: The bleSensorTag Sensor Stream object definition

Full User Report object definition(s):

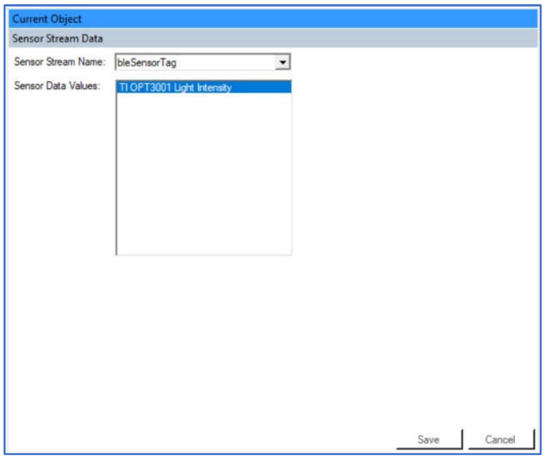
Figure 148 shows the full definition for the *sensorDataReport* UserReport object. User Report objects act as a template for the creation of custom tailored data packets. In the *Report Items* dialog box users can add and arrainge data components of their packet. Four data types are currently supported and may be added by clicking the *Add* button, (1) Sensor Stream Data values, (2) Data Averager values, (3) Data Cache values, and (4) GPIO Pin values. This example will contain four elements, the most recent SensorTag Lumen value, the most recent Tempe Temperature value, the average temperature value from the previously created Averager object, and the Cache values from the previously created Cache object. Below are the definition screens used to create the individual report components.



Name: sensorDataReport
Report ID: SensorData
Report Format: XML
Report Items:
- Sensor Stream Data: bleSensorTag (TI O
- Sensor Stream Data: antTempe (Current
- Sensor Data Average: doInitTempAverag
- Sensor Data Cache: doInitTempCache, 1

Figure 80: The sensorDataReport Report object definition

Figure 149 shows the full definition for the Sensor Stream Data value included in the *sensorDataReport* Report object. To add a Sensor Stream data value, click the *Add* button in the main *User Report Specification* screen and select Sensor Stream Data from the dropdown. All previously defined Sensor Streams will appear in the *Sensor Stream Name* dropdown. Choosing a Sensor Stream will then populate the *Sensor Data Values* multi-select dialog box with all values available from that Sensor Stream. Users may multi-select the values they wish to add. In this example the Light Intensity (lumens) data from the SensorTag Sensor Stream will added to the custom packet.

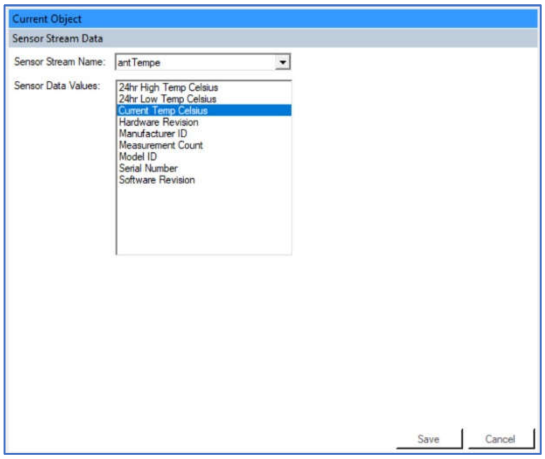


Sensor Stream Name: bleSensorTag

Sensor Data Values: TI OPT3001 Light Intensity

Figure 81: The Sensor Stream Data report element definition

Figure 150 shows the full definition for the Sensor Stream Data value added to the *sensorDataReport* Report object. To add a Sensor Stream data value, click the *Add* button in the main *User Report Specification* screen and select Sensor Stream Data from the dropdown. All previously defined Sensor Streams will appear in the *Sensor Stream Name* dropdown. Choosing a Sensor Stream will then populate the *Sensor Data Values* multi-select dialog box with all values available from that Sensor Stream. Users may multi-select the values they wish to add. In this example the Current Temp Celsius data from the antTempe Sensor Stream will added to the custom packet.

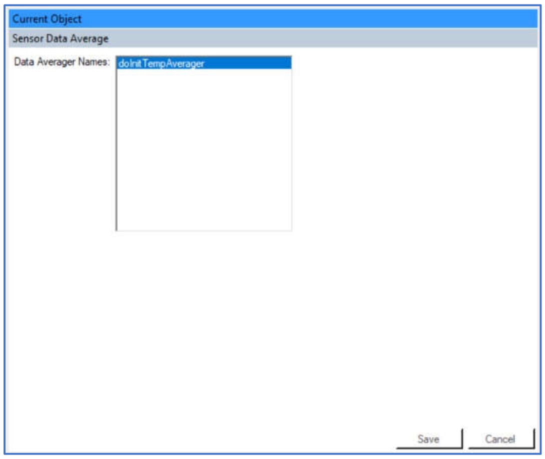


Sensor Stream Name: antTempe

Sensor Data Values:
- 24hr High Temp Celsius
- 24hr Low Temp Celsius
- Current Temp Celsius
- Hardware Revision
- Manufacturer ID
- Measurement Count
- Model ID
- Serial Number
- Software Revision

Figure 82: The Sensor Stream Data report element definition

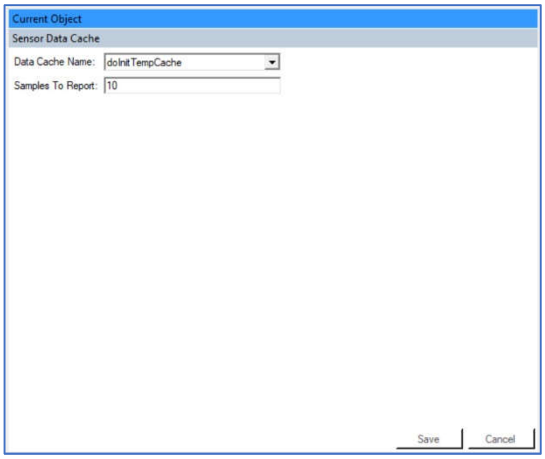
Figure 151 shows the full definition for the Sensor Data Averager value added to the *sensorDataReport* Report object. To add an Averager data value, click the *Add* button in the main User Report Specification screen and select *Sensor Data Average* from the dropdown. All previously defined Averager objects will appear in the *Data Average Names* multi-select dialog box. Users may multi-select the Averager object values they wish to add. In this example the value from the Averager object created by the *doInitTempAverager* Action object will added to the custom packet.



Data Averager Names: doInitTempAverager

Figure 83: The Sensor Data Averager report element definition

Figure 152 shows the full definition for the Sensor Data Cache values added to the *sensorDataReport* Report object. To add one or more Cache data values, click the *Add* button in the main User Report Specification screen and select *Sensor Data Cache* from the dropdown. All previously defined Cache objects will appear in the *Data Cache Name* dropdown box. Users may select the Cache object who's values they wish to add as well as the number of cached values to report. In this example the ten most recent values from the Cache object created by the *doInitTempCache* Action object will added to the custom packet.


Current Object

Sensor Data Cache

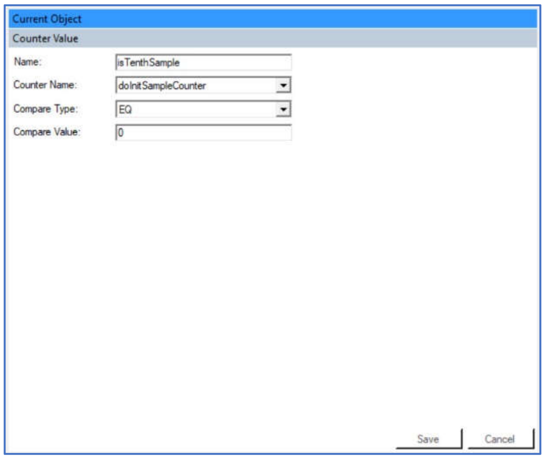
Data Cache Name: doInitTempCache

Samples To Report: 10

Figure 84: The Sensor Data Cache report element definition

Full State object definition(s):

Figure 153 shows the full definition for the *isTenthSample* CounterValue State object. When executed this object will test whether the Counter object created via the *doInitSampleCounter* Action object is equal to (EQ) the user specified value in the *Compare Value* dialog box. In this example, since the Counter object has a modulo of 10, the State object will return *true* after every tenth sample of the antTemp Temperature has been taken.


Current Object

Counter Value

Name: isTenthSample

Counter Name: doInitSampleCounter

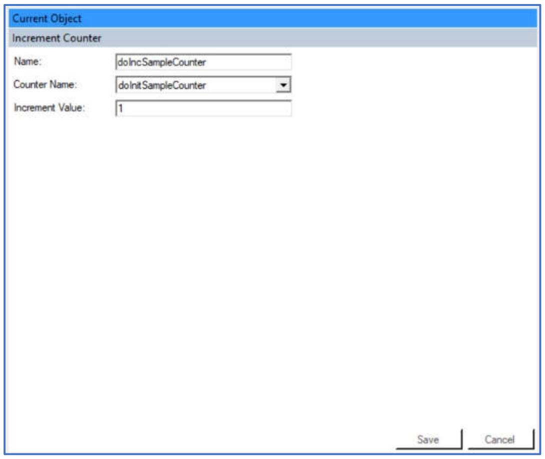
Compare Type: EQ

Compare Value: 0

Figure 85: The isTenthSample State object definition

Full Action object definition(s):

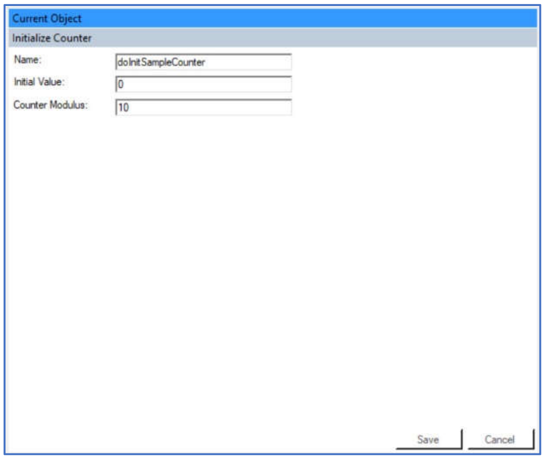
Figure 154 shows the full definition for the *doIncSampleCounter* IncrementCounter Action object. When executed this Action will increment the counter selected by the user in the *Counter Name* dropdown by specified value in the *Increment Value* text box. In this example the sample counter previously defined by the *doInitSampleCounter* Action object will be incremented by one.



Name: doIncSampleCounter
Counter Name: doInitSampleCounter
Increment Value: 1

Figure 86: The doIncSampleCounter Action object definition

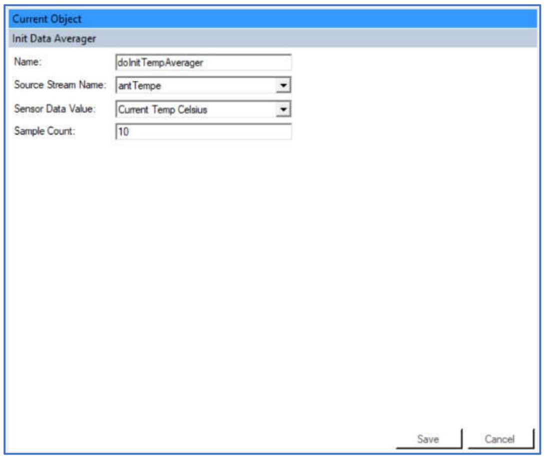
Figure 155 shows the full definition for the *doInitSampleCounter* InitializeCounter Action object. When executed this Action will create and initialize a Counter object. In this example the new Counter object will be created and initialized with an *Initial Value* of 0 and a *Counter Modulus* of 10.



Name: doInitSampleCounter
Initial Value: 0
Counter Modulus: 10

Figure 87: The doInitSampleCounter Action object definition

Figure 156 shows the full definition for the *doInitTempAverager* InitDataAverager Action object. When executed it will create a new Averager object. In this example the antTempe Sensor Stream has been selected from the *Source Stream Name* dropdown. This causes the *Sensor Data Value* dropdown to be populated with all available antTempe Sensor Stream data values. Current Temp Celsius has been selected as the value to be averaged. Lastly, a *Sample Count* of 10 indicates this is to be a moving average, reflecting the average of the last 10 values added into the Averager object via the doUpdateTempAverager Action object.



Current Object

Init Data Averager

Name: doInitTempAverager

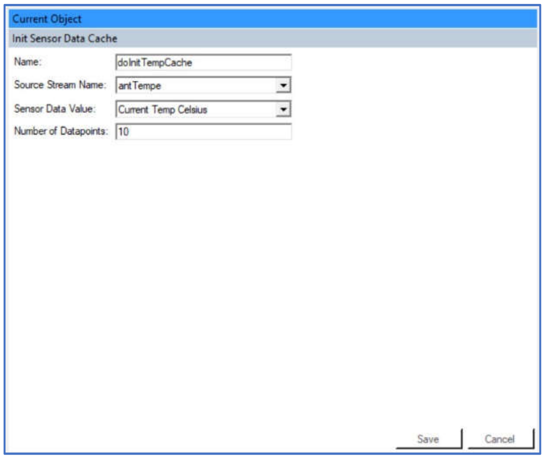
Source Stream Name: antTempe

Sensor Data Value: Current Temp Celsius

Sample Count: 10

Figure 88: The doInitTempAverager Action object definition

Figure 157 shows the full definition for the *doInitTempCache* InitSensorDataCache Action object. When executed this object will create a new Cache object. In this example the antTempe Sensor Stream has been selected from the *Source Stream Name* dropdown. This causes the *Sensor Data Value* dropdown to be populated with all available antTempe Sensor Stream data values. Current Temp Celsius has been selected as the value to be cached. Lastly, setting the *Number of Datapoints* to 10 indicates that this is a circular cache which will hold the last 10 values added.



Current Object

Init Sensor Data Cache

Name: doInitTempCache

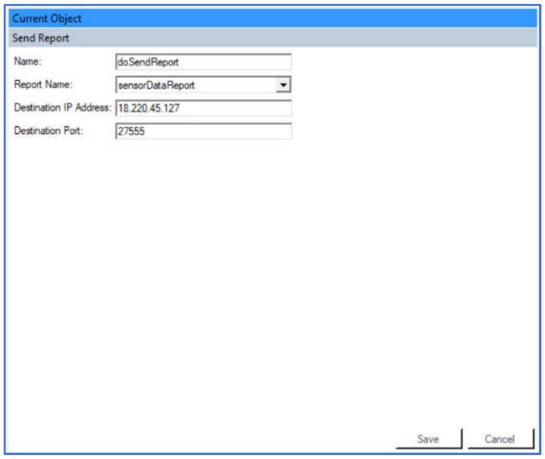
Source Stream Name: antTempe

Sensor Data Value: Current Temp Celsius

Number of Datapoints: 10

Figure 89: The doInitTempCache Action object definition

Figure 158 shows the full definition for the *doSendReport* SendReport Action object. When executed this object will assemble and send the User Report data packet chosen in the *Report Name* dropdown to the user specified IP and port address over a TLS encrypted connection. The *Destination IP Address* identifies the IP address of a remote packet capture server to forward the packets to. The *Destination Port* identifies what port the remote packet capture service has been configured to listen on.



Current Object

Send Report

Name: doSendReport

Report Name: sensorDataReport

Destination IP Address: 18.220.45.127

Destination Port: 27555

Figure 90: The doSendReport Action object definition

Figure 159 shows the full definition for the *toStartReportTimer* StartContinuousTimer Action object. When executed this object will create and start a new continuous timer object. Continuous timer objects will continually restart themselves on expiration. For this example the *Timer Duration* has been set to 30000 millseconds (30 seconds). The *Timer Expired Event* dialog box defines an optional TimerExpired object to be queued each time the timer expires. Here, every 30 seconds an *01-onReportTimerExpires* TimerExpired object will be put to the rule engine's event queue.

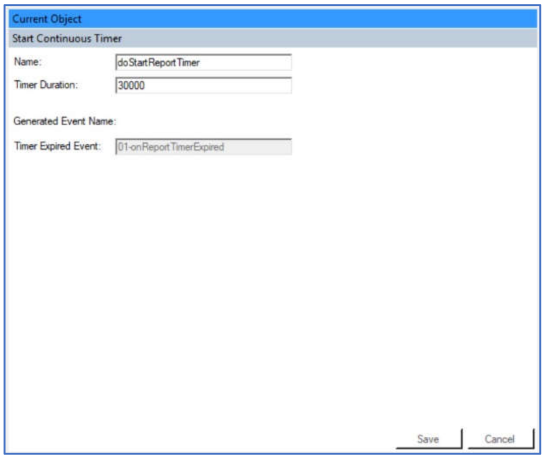

Current Object
Start Continuous Timer
Name: doStartReportTimer
Timer Duration: 30000
Generated Event Name:
Timer Expired Event: 01-onReportTimerExpired

Figure 91: The doStartReportTimer Action object definition

Figure 160 shows the full definition for the *doUpdateTempAverager* UpdateDataAverager Action object. When executed this Action object will average in the latest data value as specified in the Averager object chosen in the *Data Averger Name* dropdown. In this example, the Averager object created by the *doInitTempAverager* Action object will be updated with the latest *Current Temperature* value from the antTempe Sensor Stream.

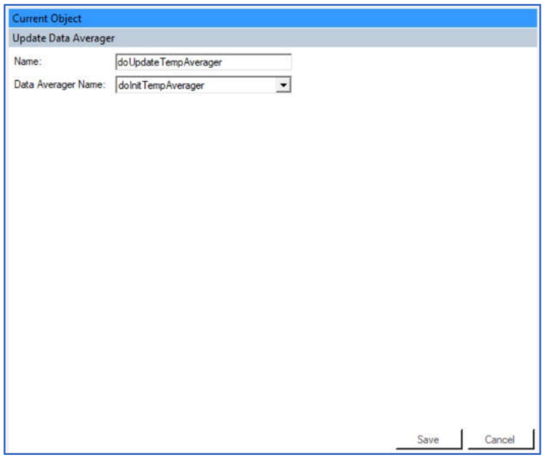

Current Object
Update Data Averager
Name: doUpdateTempAverager
Data Averager Name: doInitTempAverager

Figure 92: The doUpdateTempAverager Action object definition

Figure 161 shows the full definition for the *doUpdateTempCache* UpdateSensorDataCache Action object. When executed this Action object will cache the latest data value as specified in the Cache object chosen in the *Data Cache Name* dropdown. In this example, the Cache object created by the *doInitTempCache* Action object will be updated with the latest *Current Temperature* value from the antTempe Sensor Stream.

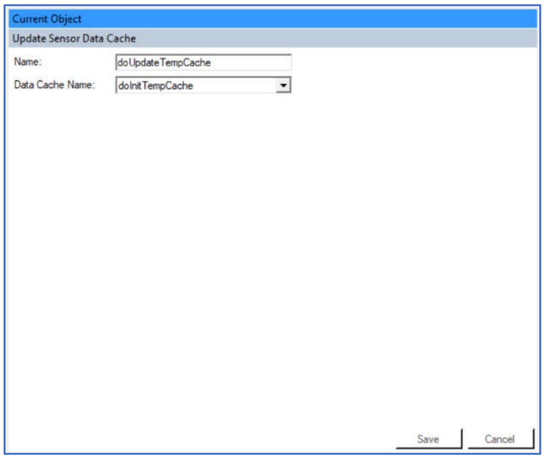

Current Object
Update Sensor Data Cache
Name: doUpdateTempCache
Data Cache Name: doInitTempCache

Figure 93: The doUpdateTempCache Action object definition

Example Summary:

Example 5 has added new objects to extend the functionality presented in Example 1 including Counter objects, Averager objects, Cache objects, and User Report objects. SoftHub rulesets can be thought of as a series of *when/if/do* statements (i.e. **WHEN** *some event occurs*, **IF** *some conditions are met*, **DO** *some action(s)*). The table below provides a summary of how the SoftHub's rule engine will process the ruleset described in Example 5:

Event (When)	**Rule**	**State (If)**	**Action (Do)**
00-onSoftHubStartup: The SoftHub application has started.	00-doSoftHubInit	N/A Do Always	doInitTempAverager: Create a 10 value moving average Averager object for the Tempe sensor's *Current Temperature* value.
As Above	As Above	As Above	doInitTempCache: Create a 10 item Cache object for the Tempe sensor's *Current Temperature* value.
As Above	As Above	As Above	doInitSampleCounter: Create a cached samples Counter object with a modulo of 10.
As Above	As Above	As Above	doStartReportTimer: Create and start a continuous Timer object that will expire every 30 seconds.
01-onReportTimerExpired: The 30 second continuous timer has expired.	00-onDataUpdates	N/A Do Always	doUpdateTempAverager: Update the Averager object with the last received *Current Temperature* data point.
As Above	As Above	As Above	doUpdateTempCache: Update the Cache object with the last received *Current Temperature* data point.
As Above	As Above	As Above	doIncSampleCounter: Increment the number of samples in the current report cycle.
As Above	00-doCheckSendReport	Do only if executing State object *isTenthSample* returns *true*.	doSendReport: Create and send a custom tailored User Report packet to the remote packet capture server.

Below is an sample packet generated by Example 5. Note the four sections under the Packet Payload field, two SensorDataValues sections (bleSensorTag/antTempe), followed by a SensorAveragedData section, and lastly the SensorCachedData section which lists each of the individual antTempe temperature values used to create the average.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<OmnIotHubPacket>
 <OriginDeviceId>555</OriginDeviceId>
 <PacketGeneratedTimestamp>06/16/2018 11:36:48 932:693 UTC</PacketGeneratedTimestamp>
 <PacketReceivedTimestamp>06/16/2018 11:36:49 147:383 UTC</PacketReceivedTimestamp>
 <SequenceNumber>1</SequenceNumber>
 <PacketType>XML Report</PacketType>
 <PacketContents>
  <OmnIotRptPacket>
   <ReportID>SensorData</ReportID>
   <SequenceNum>1</SequenceNum>
   <TimeStamp>06/16/2018 04:36:48</TimeStamp>
   <PacketPayload>
    <SensorDataValues>
     <SensorName>bleSensorTag</SensorName>
     <SensorDataField>
      <FieldName>TI OPT3001 Light Intensity</FieldName>
      <TimeStamp>06/16/2018 04:36:48</TimeStamp>
      <FieldValue>2.75</FieldValue>
     </SensorDataField>
    </SensorDataValues>
    <SensorDataValues>
     <SensorName>antTempe</SensorName>
     <SensorDataField>
      <FieldName>Current Temp Celsius</FieldName>
      <TimeStamp>06/16/2018 04:36:46</TimeStamp>
      <FieldValue>29.2999992371</FieldValue>
     </SensorDataField>
    </SensorDataValues>
    <SensorAveragedData>
     <SensorName>antTempe</SensorName>
     <FieldName>Current Temp Celsius</FieldName>
     <AveragedValue>29.9559997559</AveragedValue>
    </SensorAveragedData>
```

Continued on the following page…

```
    <SensorCachedData>
     <SensorName>antTempe</SensorName>
     <FieldName>Current Temp Celsius</FieldName>
     <CachedDataValues>
      <DataPair>
       <TimeStamp>06/16/2018 04:36:46</TimeStamp>
       <DataValue>29.2999992371</DataValue>
      </DataPair>
      <DataPair>
       <TimeStamp>06/16/2018 04:36:18</TimeStamp>
       <DataValue>29.2999992371</DataValue>
      </DataPair>
      <DataPair>
       <TimeStamp>06/16/2018 04:35:46</TimeStamp>
       <DataValue>29.7600002289</DataValue>
      </DataPair>
      <DataPair>
       <TimeStamp>06/16/2018 04:35:18</TimeStamp>
       <DataValue>29.7600002289</DataValue>
      </DataPair>
      <DataPair>
       <TimeStamp>06/16/2018 04:34:48</TimeStamp>
       <DataValue>29.7600002289</DataValue>
      </DataPair>
      <DataPair>
       <TimeStamp>06/16/2018 04:34:16</TimeStamp>
       <DataValue>29.7600002289</DataValue>
      </DataPair>
      <DataPair>
       <TimeStamp>06/16/2018 04:33:48</TimeStamp>
       <DataValue>30.8199996948</DataValue>
      </DataPair>
      <DataPair>
       <TimeStamp>06/16/2018 04:33:16</TimeStamp>
       <DataValue>30.8199996948</DataValue>
      </DataPair>
      <DataPair>
       <TimeStamp>06/16/2018 04:32:48</TimeStamp>
       <DataValue>30.1399993896</DataValue>
      </DataPair>
      <DataPair>
       <TimeStamp>06/16/2018 04:32:16</TimeStamp>
       <DataValue>30.1399993896</DataValue>
      </DataPair>
     </CachedDataValues>
    </SensorCachedData>
   </PacketPayload>
  </OmnIotRptPacket>
 </PacketContents>
</OmnIotHubPacket>
```

Addendum A: MQTT Extended Options Files (.mcfg)

A.01 Introduction

The MQTT Standard describes a set of parameters that users may provide to dictate how connections between clients and MQTT brokers behave. The SoftHub MQTT implementation is built on top of the paho library, and as such all standard parameters are exposed for user control. Due to the number of options available, they are exposed in the SoftHub as sets of “basic” and “extended” options. Basic options can be set directly in the SoftHub Configuration Utility when defining a new *PublishData* or *PublishReport* Action object as part of a rule set. Both the *PublishData* and the *PublishReport* object definition screens provide a field for the user to optionally specify an “Extended Options File”. These files allow users to set the extended options for an MQTT connection not exposed in the SoftHub Configuration Utility directly.

During installation, template MQTT Extended Options files are copied to the configuration file directory where the RuleEngine.xml file resides for a specific platform. User modified Extended Options files should also be kept only in this directory. To create a new file, copy the template file to the filename you specified in the SoftHub Configuration Utility (note however that in the Configuration Utility you don’t specify the .mcfg suffix). To enable a parameter you wish to set, remove the leading pound sign (comment indicator). Note also that if a parameter is set both in the SoftHub Configuration Utility and in the Extended Options file, the former will take precedence and the latter will be ignored. Be sure to read the comments in the template file to ensure you specify your options in the proper format. **Enabled Extended Options file parameters must be properly formatted or all settings will be ignored**.

The following sections detail the parameters available in each subsection of an Extended Options file. Note that these are only very high level descriptions. For specific behaviors of individual parameters please reference the MQTT standard. All parameters not denoted with an asterisk come directly from the standard (whereas asterisk’ed parameters are SoftHub specific).

A.02 General Connection Settings

The first subsection in an Extended Options file include the general connection settings. Note that, as detailed above, if values in this section conflict with those set in the RuleEngine.xml file, they will be ignored and the value set in the RuleEngine.xml file will take precedence. The table below details the connection parameters that can be enabled and set in this subsection –

ConnectionUsername	The User Id to be passed when connecting to the remote MQTT broker
ConnectionPassword	The User Password to be passed when connecting to the remote MQTT broker
ConnectionClientId	The Client Id to be passed when connecting to the remote MQTT broker
ConnectionAddress	The remote address to be used when connecting to the MQTT broker
DeviceControlTopic	A topic to subscribe to to receive User Defined Event trigger directives
ConnectionKeepAlive	The MQTT heartbeat keep alive interval in seconds
ConnectTimeout	The number of seconds to wait for a connect/publish/disconnect operation before considering it as having timed out.
MqttVersion	The MQTT API version to use when communicating with the remote server
FlattenJson(*)	Setting this value to "1" will enable JSON flattening. Some MQTT brokers may disallow label/value pairs being nested beyond the 1st level.

A.03 Last Will Connection Settings

Any desired Last Will and Testament settings desired can be set in this subsection. See the MQTT standards documentation for details on how these parameters may interact -

LastWillTopic	Topic for publishing "last will" message
LastWillMsg	"Last will" message to be published on unexpected connection drop
LastWillRetained	Flag (0=false/1=true) indicating whether the last will message should be retained on the broker

A.04 SSL/TLS Connection Settings

This subsection allows users to control advanced settings relating to SSL/TLS encrypted connections. Your target MQTT broker may require the use of specific settings to ensure proper security. Also see the MQTT standards documentation for details on how these parameters may interact -

SslTrustedStore	The PEM format file containing the public digital certs trusted by the client
SslTrustedKey	The PEM format file containing the public cert chain of the client
SslPrivateKey	If not included in the SslTrustedKey store, points to the PEM format file containing the client's private key
SslPrivateKeyPwd	The password to load the client's privateKey if encrypted
SslCypherSuites	List of ciphers the client will present to the server during SSL handshake
SslEnableCertAuth	Flag (0=false/1=true) to enable verification of the server certificate
SslVersion	The SSL/TLS version to use
SslPostVerify	Flag to carry out post-connect checks, including that a certificate matches the given host name

A.05 JSON Filter-Format Files

JSON Filter/Format Files are used to reformat flattened JSON packet data on the fly. Each file will create a single filter/format object and each object will be applied to each new packet in the order specified. As soon as a match is found, that Filter/Format transformation will be applied and no further processing will be performed. JSON Filter/Format files are covered in detail in the following section.

PublishUnformatted(*)	Determines whether packets that are unfiltered by any of the active filters are published or discarded
JffFile01(*)	The name of the first JSON Format/Filter file to be applied to new outgoing packets
.	
.	
JffFile16(*)	The name of the last JSON Format/Filter file to be applied to new outgoing packets

A.06 External Control Via User Defined Events

When defining a new *PublishData* or *PublishReport* Action object, you have the option of specifying a topic for the SoftHub to listen to for incoming control messages. These messages can be used to queue any User Defined Events defined in the active configuration ruleset.

The SoftHub allows for two methods of mapping incoming MQTT control data messages to User Defined Events. The first is to simply prefix the name of the User Event you want to queue with a "UEV_" tag and include that string in the message sent from the MQTT broker. If the string is found anywhere in the incoming text it will be mapped accordingly. For MQTT brokers that do not allow you to modify the outgoing control data, the method below allows you to map any string, or string fragment, in the MQTT control message to a User Defined Event.

This subsection allows users to specify "Input to User Event" mappings. Each of the 16 available slots allows you to map input from any subscribed topic to your own internal User Defined Events. Each entry should follow the format of a quoted string that acts as a search string, followed immediately by a comma, followed immediately by a quoted string matching the name of a User Defined Event you wish to queue and have created in the active RuleEngine.xml file. Note that search strings allow for minimal wildcarding - you may prefix your string with an asterisk indicating your pattern must match the end of a string, you may suffix your string with an asterisk indicating your pattern must match the beginning of a string, or you may prefix and suffix your string with an asterisk indicating your pattern can match any substring in a string. See the .jff template file for examples of how to properly format these fields.

As will JSON Filter/Format objects, the SoftHub will process each mapping in order until a match is found at which time the transformation will be made and no further processing will occur.

InputToUEV01(*)	The first pattern/replacement pair to be applied to incoming control messages.
.	
.	
InputToUEV16(*)	The last pattern/replacement pair to be applied to incoming control messages.

Addendum B: JSON Filter-Format Files (.jff)

B.01 Introduction

Like the objects you define in the SoftHub Configuration Utility, JSON Filter/Format files map directly to objects that actually perform a specific function. In this case, the objects filter outgoing JSON packets to identify target packets to be re-formatted to meet the requirements of some target MQTT broker. By default, the SoftHub will provide all information in a binary packet fully decoded into a JSON packet. Since this format will not be suitable as input to many third party MQTT brokers, JFF objects allow you to selectively pull the data from your packets and insert the selected data in any packet format you define. Using this method, the SoftHub should able to send data directly to most, if not all, third party MQTT brokers.

Each JFF file can perform one type of transformation and you may specify up to sixteen JFF files per MQTT connection. There are three types of parameters to be defined in a JFF file. They are described in detail in the following sections, but in general they are (1) filter parameters that allow you to selectively target specific packets for transformation, (2) target value labels which identify the labels associated with the values you want to insert into your transformed packet, and (3) an output format specifier which acts as a template for the packet you wish to actually publish.

Note that to use JFF files your packets must be in flattened JSON format. That is to say that JSON must be the format you selected in the *PublishData* or *UserReport* object definition screen, and that the "*FlattenJson*" option must be set to "1" in the associated .mcfg file.

B.02 Pattern Match Specifiers

On a given MQTT connection you may have multiple packet formats being published at any given time. You may have multiple sensors connected and each sensor may produce one or more packet formats. Additionally, you may define one or more custom report packets to publish control or aggregated data from multiple sensors as well. The "Pattern Match Specifiers" subsection of a JFF file is where you can single out specific label/value pairs to ensure that the packet you attempt to transform is the type of packet you are targeting. While Pattern Match Specifiers are optional, you can specify up to eight per JFF file. Each specifier consists of a label/value pair that must all be matched for the packet to be further processed. See the template JFF file for the exact format of Pattern Match Specifiers and for examples of how these strings should be defined.

B.03 Data Label Specifiers

Assuming all the Pattern Match Specifiers are matched, the second subsection of a JFF file is where you define the labels of the data fields you wish to transfer to your reformatted output packet. The important thing to understand is that the labels are coupled to the Output Format Specifier (detailed in the next section). Each Data Label Specifier will indicate one data value to be inserted into the output packet and you can specify up to sixteen labels per JFF file. It should be noted that, like Pattern Match Specifiers, if all labels are not found no transformation will be applied and processing will move to the next JFF file (if additional JFF files are defined).

Data Label Specifiers also may serve a secondary function wherein the data value itself may be reformatted as well. See the JFF template comments for details but current data transformations are as follows -

"1" - Transform an ISO 8601 date string to a UNIX epoch date string value with a 1 sec resolution
"2" - Transform an ISO 8601 date string to a UNIX epoch date string value with a 1 msec resolution

B.04 Output Format Specifier

Each JFF File must have exactly one Output Format Specifier defined. The Output Format Specifier acts as the template for the reformatted output packet. The process is very straight-forward, wherein data values associated with the labels defined via the Data Label Specifiers are substituted directly in the Output Format Specifier where ever a "%s" value placeholders occur. See the JFF template file for examples but the important things to understand are that -

(1) the values will be replaced from left to right by the lowest to the highest numbered Data Label Specifier
(2) the number of "%s" value place holders must exactly match the number of Data Label Specifiers
(3) there may be no gaps in the Data Label Specifiers, all must be consecutive.

Addendum C: SoftHub Object Tables

C.01 Sensor Stream Object(s) Table

The table below provides a synopsis of all currently supported *Sensor Stream* object(s) –

Object Name	**Description**
AntSensorStream	Used to define an ANT sensor to be connected to the SoftHub.
BleSensorStream	Used to define a BLE sensor to be connected to the SoftHub.

C.02 User Report Object(s) Table

The table below provides a synopsis of all currently supported *User Report* object(s) –

Object Name	**Description**
UserReport	Used to create a customized report packet template, in either binary, JSON, or XML format, aggregating and consolidating data from multiple sources.

C.03 Event Object(s) Table

The table below provides a synopsis of all currently supported *Event* object(s) –

Object Name	Description
AppStarting	Always queued by the SoftHub as the first event to be processed.
AppStopping	Queued either internally or externally to initiate all final processing and exit from the SoftHub Application.
UserDefinedEvent	User Defined Events are defined by the user and can only be queued in response to incoming MQTT control messages.
SensorConnect	Sensor Stream object specific, queued when the associated sensor has successfully connected to the SoftHub.
SensorDisconnect	Sensor Stream object specific, queued when the associated sensor has disconnected (either expectedly or unexpectedly) from the SoftHub.
NewSensorPacket	Sensor Stream object specific, queued every time a new packet has been received from the associated sensor.
GpioPinLow	GPIO pin specific, queued when an associated GPIO input pin has transitioned from a *high* to a *low* state.
GpioPinHigh	GPIO pin specific, queued when an associated GPIO input pin has transitioned from a *low* to a *high* state.
TimerExpired	Timer object specific, queued when the associated timer has elapsed.

C.04 Action Object(s) Table

The table below provides a synopsis of all currently supported *Action* object(s) –

Object Name	**Description**
EnableSensorStream	Sensor Stream object specific, directs the SoftHub to attempt to connect to the associated sensor.
DisableSensorStream	Sensor Stream object specific, directs the SoftHub to initiate a disconnect from the associated sensor.
EnableBleNotifications	Enable a Ble sensor characteristic to begin notifications.
DisableBleNotifications	Disable a Ble sensor characteristic from continuing notifications.
InitiateBleRead	Initiate an asynchronous read from a Ble sensor characteristic.
InitiateBleWrite	Initiate an asynchronous write to a Ble sensor characteristic.
InitDataAverager	Creates and initializes an "averager object" for a specific sensor value. May be a moving or running average. These objects may be included in *UserReport* Report objects, or may be used as threshold values in *SensorDataAveragerValue* State objects to conditionally control the execution of Rule objects.
UpdateDataAverager	Adds a new data point to an averager object previously created via a *InitDataAverager* Action object.
InitSensorDataCache	Creates and initializes a circular "sensor data cache object" for a specific sensor value. These objects may be included in *UserReport* Report objects, or may be used as threshold values in *NumCachedSamplesValue* State objects. NumCachedSamplesValue State objects may be used to conditionally control the execution of Rule Objects.
UpdateSensorDataCache	Adds a new data point to sensor data cache object previously created via a *InitSensorDataCache* Action object.
SendData	Sensor Stream object specific, forwards the most recently received packet from the associated sensor to a remote packet capture server.
SendReport	Causes the associated *UserReport* object packet to be assembled and forwarded to a remote packet capture server.
PublishData	Sensor Stream object specific, publishes the most recently received packet from the associated sensor to a user specified remote MQTT broker.
PublishReport	Causes the associated *UserReport* object packet to be assembled and published to a user specified remote MQTT broker.
LogData	Sensor Stream object specific, logs the most recently received packet from the associated sensor to the SoftHub device's internal storage.
LogReport	Causes the associated *UserReport* object to be assembled and logged to the SoftHub device's internal storage.
SendEmail	Will cause the SoftHub to attempt to send an email or SMS message to one or more specified recipients.
StartOneShotTimer	Will cause the creation/start of a "one-shot" (non-repeating) timer object expiring at some relative time in the future (e.g. 100 milliseconds). On expiration, an associated *TimerExpired* Event object will be put to the rule engine's event queue.
StartContinuousTimer	Will cause the creation/start of a "continuous" (repeating) timer object expiring at some relative time in the future. On expiration, an associated *TimerExpired* Event object will be put to the rule engine's event queue.

StartOneShotTodTimer	Will cause the creation/start of a one-shot "time of day" timer object expiring at some absolute time in the future (e.g. 4:30 PM). On expiration, an associated *TimerExpired* Event object will be put to the rule engine’s event queue.
StartContinuousTodTimer	Will cause the creation/start of a continuous "time of day" timer object expiring at some absolute time in the future. On expiration, an associated *TimerExpired* Event object will be put to the rule engine’s event queue.
StopTimer	Timer specific, will cancel a previously created/started timer object (i.e. no *TimerExpired* Event object will be queued).
InitializeFlag	Will cause the creation/initialization of a Boolean Flag object. Flag objects may be referenced by *FlagState* State objects to conditionally control the execution of Rule objects.
SetFlag	Sets a Flag object created by an *InitializeFlag* Action object to "true".
ClearFlag	Sets a Flag object created by an *InitializeFlag* Action object to "false".
ToggleFlag	Toggles the value of a flag object created by an *InitializeFlag* Action object.
InitializeCounter	Will cause the creation/initialization of a Counter object. Counter objects may be referenced by *CounterValue* State objects to conditionally control the execution of Rule objects.
IncrementCounter	Will increment a counter created by the *InitCounter* Action object by a user defined value.
DecrementCounter	Will decrement a counter created by the *InitCounter* Action object by a user defined value.
SetGpioPinLow	Will set an output GPIO pin that has been enabled in the SoftHub's GPIOs System Options to "low".
SetGpioPinHigh	Will set an output GPIO pin that has been enabled in the SoftHub's GPIOs System Options to "high".
ToggleGpioPin	Will toggle an output GPIO pin that has been enabled in the SoftHub's GPIOs System Options to its inverted state.
EnableHubAction	Will reenable execution of a specific Action object that has been previously disabled.
EnableHubEvent	Will reenable the processing of a specific Event object that has been previously disabled.
EnableHubRule	Will reenable the evaluation of a specific Rule object that has been previously disabled.
DisableHubAction	Will disable execution of a specific Action object by any rule that references it.
DisableHubEvent	Will disable the processing of a specific Event object by the SoftHub rule engine causing all associated Rule objects to *not* be evaluated.
DisableHubRule	Will disable the evaluation of a specific Rule object by the SoftHub rule engine when its associated Event object is received.
ToggleHubAction	Will toggle the enabled/disabled flag of a specific Action object.
ToggleHubEvent	Will toggle the enabled/disabled flag of a specific Event object.
ToggleHubRule	Will toggle the enabled/disabled flag of a specific Rule object.
ExecSysCommand	Executes a system command as a background task.
SoftHubStop	Will cause an *AppStopping* Event object to be written to the rule engine event queue, which in turn will cause the SoftHub Application to initiate its own shutdown and exit.

C.05 State Object(s) Table

The table below provides a synopsis of all currently supported *State* object(s) –

Object Name	**Description**
SensorConnected	Sensor Stream object specific, when executed this object will be *true* if the associated sensor is currently connected to the SoftHub.
SensorDataValue	When executed this object will test a specific sensor data value against a user defined threshold value.
SensorDataAveragerValue	When executed this object will test a specific *InitDataAverager* Action object's current value against a user defined threshold value.
NumCachedSamplesValue	When executed this object will test a specific *InitSensorDataCache* Action object's cached sample count against a user defined threshold value.
FlagState	When executed this object will test a specific Flag object value against a user defined compare value.
CounterValue	When executed this object will test a specific Counter object value against a user defined threshold value.
GpioPinState	When executed this object will test a current input GPIO state against a user defined compare value.
TimeOfTheDay	When executed this object will test the current time of day against a user defined time of day value.
DayOfTheWeek	When executed this object will test the current day of the week against a user defined day of the week value.
DayOfTheMonth	When executed this object will test the current day of the month against a user defined day of the month value.
CurrentMonth	When executed this object will test the current month against a user defined month value.
CurrentYear	When executed this object will test the current year against a user defined year value.

C.06 User Rule Object(s) Table

The table below provides a synopsis of all currently supported *User Rule* object(s) –

Object Name	**Description**
UserRule	This object will specify a single Event object that will trigger its evaluation, an optional series of one or more State objects connected by Boolean operators to control its execution, and one or more Action objects to be executed provided the conditional logic has been met (or has been omitted).