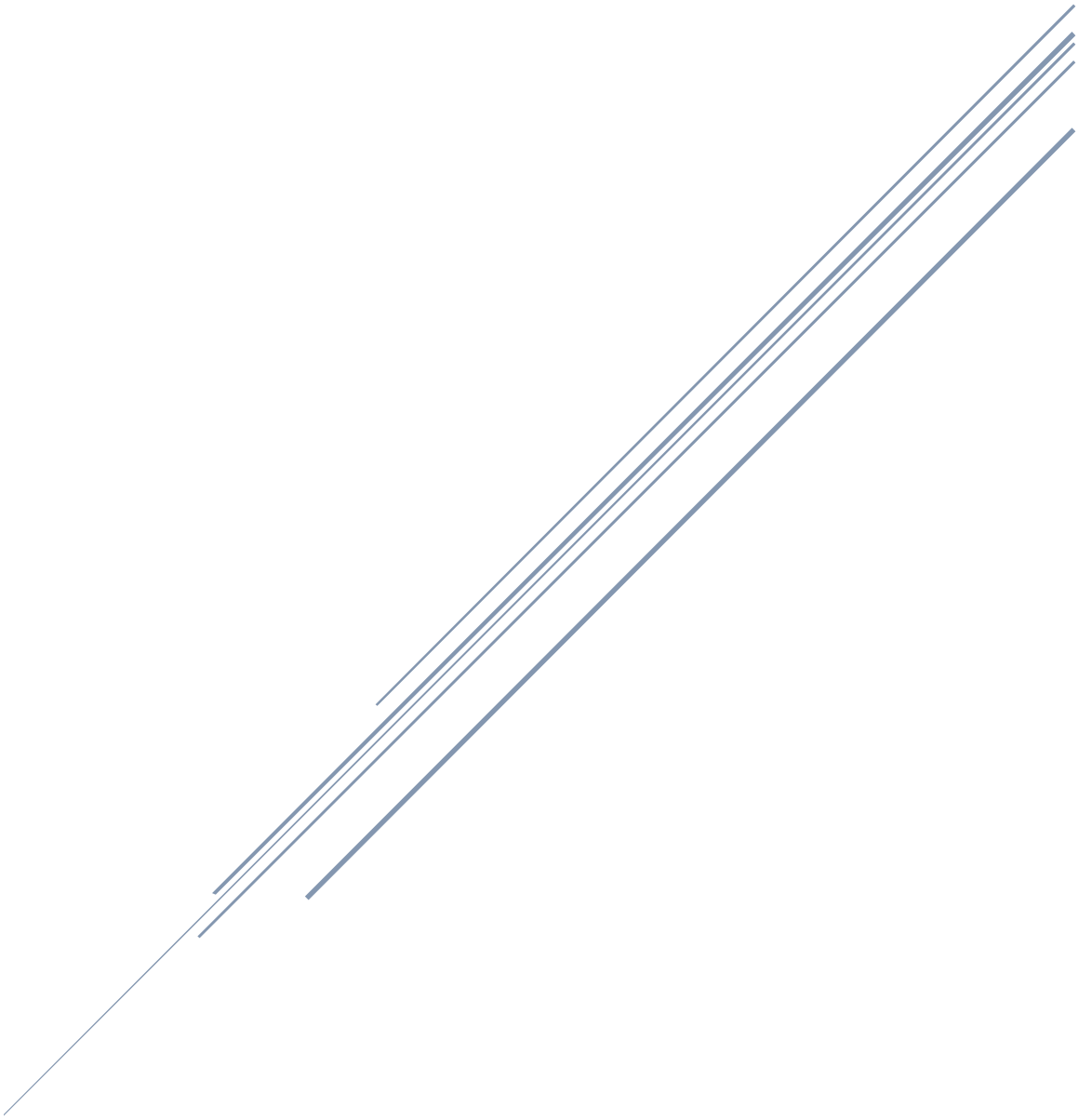
INTRODUCTION

The OmnIoT SoftHub Platform

Rel200515



www.omniot.com

Table of Contents

1. Introduction

This document provides an overview of the complete OmnIoT SoftHub Platform and each of its components. Users unfamiliar with the platform should start here before jumping into the companion document "Configuring the OmnIoT SoftHub Rule Engine".

1.1 Background Story / Project Goals

Many IoT applications will involve a common set of features when it comes to collecting and processing remote sensor data. Sensor data may be simply pushed to a cloud server to be logged and further analyzed remotely, or it may be processed locally to actuate some nearby device when exception conditions are detected, or some applications may require a combination of both. Nonetheless, until recently the process of building IoT application infrastructure at the sensor/hub data collection level was an expensive, arduous, and time consuming task. While new dedicated hub hardware devices and software API's have become available recently to aid in the process, there is still a high degree of effort involved to realize even the most basic IoT applications.

The OmnIoT SoftHub Platform has been built specifically to address the most common IoT use cases while saving the user from "reinventing the wheel", i.e. rewriting large volumes of custom code for each new application. It does this by providing at its heart a highly *configurable* "rule engine". A companion rule engine configuration utility application allows the user to define sets of objects that in turn are used to build rules which control the hub device much the way writing a custom application would. The primary difference being that where a custom application approach typically requires a team of experts to construct, the SoftHub Application can be configured to do many similar tasks in literally a matter of minutes.

An additional goal of the project is to allow the hub platform to run on a variety of hardware and software platforms. Over eighty percent of the SoftHub source code is completely platform independent ANSI standard C++. When moving to a new platform only the remaining twenty percent may need to be ported. Currently the SoftHub runs on either Debian Linux or Windows platforms. Specifically, much of the initial development has been done on the $10 Raspberry Pi Zero W.

Lastly, the rule engine has been designed to be highly extendable. The SoftHub's many subcomponents are architected to be very "loosely coupled". This should allow the platform to be easily extendable to include new wireless sensor technologies, additional external device control, cellular communications, etc., in the foreseeable future.

1.2 Platform Components

Currently there are three primary components making up the SoftHub Platform.

The SoftHub Application – This code runs on the hub device itself. When the SoftHub application starts up, it reads a user created configuration file which its internal rule engine will continually execute in the background. 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 the Windows PC application the user runs to build the configuration file that is to be read/executed by the SoftHub Application. User’s create a “RuleEngine.xml” file using the SoftHub Configuration Utility which then is distributed to one or more hub devices running the SoftHub Application. 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 used by the rule engine itself at runtime.

The Remote Packet Capture Application – Many applications require data to be forwarded to one or more central server(s) for aggregation and further processing. These central servers may physically reside remotely in the cloud or locally as part of the users' own Intranet infrastructure. Regardless, the Remote Packet Capture Application is provided to run on these servers to capture 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 however a similar Linux Daemon is planned in the future. As packets are received and decoded, a user supplied 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.

Similarly a companion utility, the "OmnIoTLogDecoder" line command, is also provided. This utility allows for the decoding of any binary log files created directly on the SoftHub itself. As with the Packet Capture Application, the Log File Decode utility ultimately passes the decoded packets to a user created DLL for processing and analysis.

1.3 Suitable Use Cases

The OmnIoT SoftHub Platform is best oriented to “Star” or “Spoke and Hub” IoT application topologies. Below are a few typical topologies that are well suited to the SoftHub.

The example labeled *Figure 1* depicts a simple self-contained sensor network configuration where multiple sensors feed into a single hub device. The hub device monitors the incoming sensor data and when it detects an exception condition it may actuate one or more physically connected devices (in this example visual or audible alarm devices). Additionally, the hub may log sensor data (in whole or in part) to its own local storage for later retrieval.

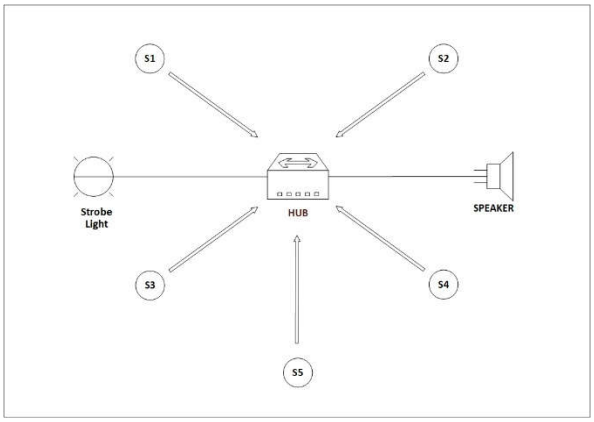


Figure 1: Simple Self Contained Sensor Configuration

These types of applications can easily be configured to be performed by the SoftHub application. Users can deploy complex logic directly in the SoftHub to detect exception conditions indicated by sensor data thresholds. Once exception conditions are detected, attached devices can be actuated directly with no user intervention required.

The sensor network in *Figure 2* illustrates a typical scenario where a suite of sensors report to a hub device which in turn is connected to a local router (either wired or wirelessly). Via the router, the hub device connects to a remote cloud server and forwards the sensor data to be logged and analyzed in the cloud. In this simplified example we have a single hub device connecting to the local router however in practice it is quite common to have multiple hub devices connecting to a single router and multiple routers feeding into one or more remote servers.

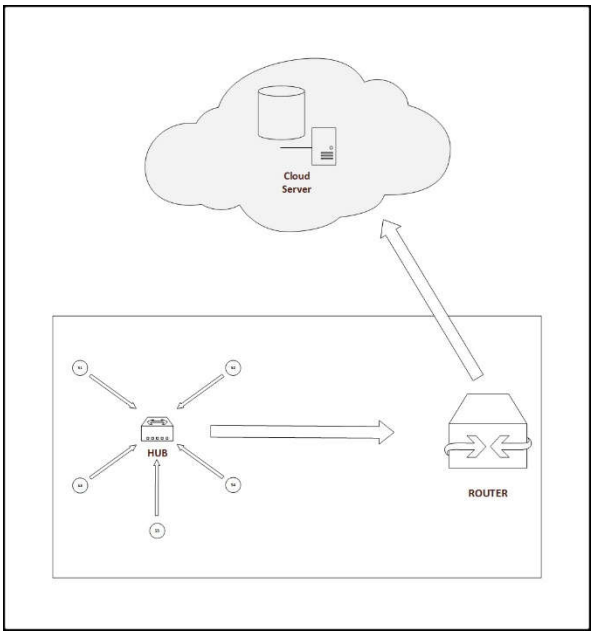


Figure 2: Sensor Data Being Forwarded to Cloud Server

Again, these types of applications can be configured in via the SoftHub in a matter of minutes. Forwarding data to one or more remote cloud servers is one of the simplest scenarios to implement. Additional data filtering can also easily be configured such that only data of interest is forwarded at some user defined interval. The SoftHub also employs an internal caching system so that in the event of a connection

outage no data will be lost. In this example typically every hub would be running the SoftHub Application and the cloud server would be running the Remote Packet Capture Application to decode the packets as they arrive.

Figure 3 illustrates a more complex sensor application topology example where multiple sensor networks feed into multiple sensor hubs. Some sensor hubs may simply forward data to one or more remote packet capture servers. Other sensor hubs may also have physically connected hardware that will be actuated locally when exception conditions are detected as the hub analyzes the incoming sensor data.

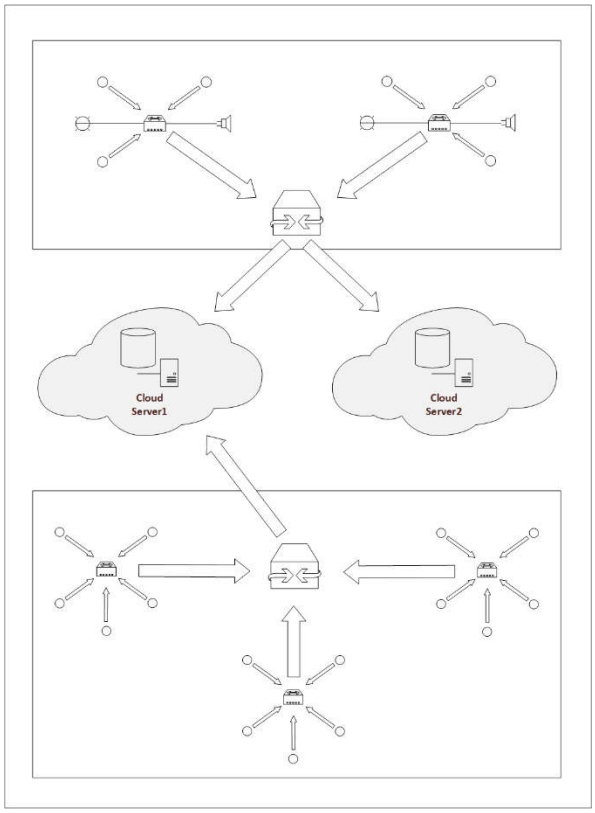


Figure 3: Moderately Complex IoT Application

In the example, multiple sensor hubs at different physical locations are connecting to one or more cloud servers through their local routers. This example is more realistic in terms of actual IoT deployments with some hub devices carrying out very similar roles while others perform more or less "one off" functions. These are exactly the types of applications the SoftHub excels at and provides the most value. Rather than writing custom code to run in each separate hub device, the user deploys hub devices running the SoftHub Application only providing a configuration file for each. Some configuration files may be unique while others may be common depending on the role the hub device may fulfill. Later sections will detail how applications are configured, but generally any of the preceding three described scenarios can be configured and deployed in a matter of minutes via the SoftHub Configuration Utility.

In building the rule engine into the SoftHub a balance has been struck between flexibility and ease of use. The rule engine is very flexible but nonetheless as a tradeoff for ease of use there will no doubt be instances where its capabilities will fall short of specific application requirements. As previously stated the primary goal is to be able to meet many of the most common IoT scenarios quickly and cheaply without requiring any custom programming as illustrated in the examples above. A following section ("Features") details some of the most common IoT hub application functionality the SoftHub can fulfill. On the other hand, some types of IoT applications that the OmnIoT SoftHub Platform is not suitable for presently include mesh applications or applications where a good deal of user input or interaction is required. These shortcomings will hopefully be addressed in a future release.

1.4 Features

The SoftHub has a core set of built in features allowing it to mimic many custom applications. By allowing the user to create their own *objects* and build their own *object based rulesets*, the SoftHub's capabilities are quite extensive. Specifically, some of the features which can be easily implemented via the configuration utility are listed below:

- Mix/match different sensors as well as different sensor technologies
- Publish sensor data directly to most third party MQTT capable cloud dashboards
- Stream TLS encrypted sensor data (either wholly or filtered) to one or more remote servers
- Similarly, log sensor data to the hub device itself when no network is available or desired
- Custom tailor your own report packets aggregating data from multiple sources
- Auto-cache data during network outages for zero packet loss
- Actuate locally attached devices via general purpose I/O's
- Capture general purpose I/O input events (e.g. button input)
- Send electronic alarms directly from the hub device via email and/or SMS
- Actuate local alarms via attached hardware (sirens, strobes, etc.)
- Cache historic data to be sent/logged only during exception conditions
- Monitor any sensor stream data value directly in the hub for threshold exceptions
- Further control any behavior via a multitude of timers (e.g. one-shot, continuous, time of day)

Note that this is not an exhaustive list, but is instead meant to provide a general sense of some of the most common IoT application requirements that can easily be configured.

1.5 Benefits

The main benefits of the OmnIoT SoftHub are its flexibility, ease of use, and low cost. In short many typical IoT applications can be fully realized in a matter of days. Similarly, changes and refinements to existing applications can be easily made as requirements evolve. Below we list some of the typical organizations that may benefit by leveraging on the SoftHub's feature set:

For New Solutions Developers - Have an idea for the next big IoT application? Need to get it built as quickly and as cheaply as possible? The OmnIoT SoftHub can jumpstart the process and save you both precious time and money. The cost of developing custom sensor/hub solutions from scratch can easily run into millions of dollars when considering software and hardware development costs (let alone long term maintenance costs). Add on the additional 6-12 month time to market cost and the inherent risk of potential engineering missteps and the benefits of an off the shelf solution become apparent. With the OmnIoT SoftHub you can have your solution in hand in literally weeks as opposed to months or more.

For Existing Solutions Providers - Already have a product or suite of IoT products in the market? We want to provide existing solution providers with the ability to extend their offerings reach with little or no additional investment. As an example, many commercially available products reply on their customer's cell phone to run an app to gather their sensor data. Adding the SoftHub into the mix allows

you to cheaply add the option of gathering data continuously, without relying on user involvement. In addition to more consistent monitoring this enables you to create your own "big data" repositories to extract the analytics needed to better meet your customers' actual needs.

For Custom Solutions Providers - Custom solutions providers and system integrators can benefit from the SoftHub much the way new solutions developers can. Lowering your bid costs and delivery time can often make the difference between winning a project bid and coming up empty handed. Our platform is built for flexibility and extensibility and most importantly *rapid deployment*. If there is a device or technology we don't support that you need, contact us as it may already be on our roadmap or easily added.

2. The SoftHub Application

The OmnIoT SoftHub Application runs as a Daemon (Linux) or Service (Windows) on the hub device, reading the user created rule engine configuration file, *RuleEngine.xml*, at startup and continuously executing the specified ruleset in the background. This section will give a general overview of how the SoftHub is built, what subcomponents are currently available, and how the Rule Engine processes the user defined rulesets.

2.1 Architecture

Architecturally, the SoftHub exists a set of "loosely coupled" subcomponents tied together and orchestrated by one central subcomponent, the *Rule Engine* subcomponent. A user defined ruleset will contain a group of objects which the Rule Engine will instantiate at startup. These objects generally exist as individual stand-alone executables. As events are detected the rule engine will execute any associated *Rule Objects* which in turn may evaluate (execute) a list of *State Objects*, which depending on their outcome may trigger the execution of one or more *Action Objects*. Each of these Action objects in turn may interact with any of the SoftHub's subcomponents to carry out its specific task.

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 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 timer event which would be captured by the SoftHub's Rule Engine subcomponent. The Rule Engine would then evaluate/execute any rules the user had associated with this event. In this example the user would have created a "Connect BLE Sensor" Action object. That object would in turn register with the SoftHub's *BLE Management Subcomponent* to request that the specified sensor be connected.

In the above simple example we have a few user created 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 Management subcomponents). Lastly, we have the Rule Engine subcomponent at the heart of everything creating the objects and orchestrating their execution. The following section details each of the currently available SoftHub subcomponents. Later sections will describe each of the rule engine objects currently available.

2.2 Application Subcomponents

This section provides an overview of the currently available SoftHub subcomponents. It is provided solely as a reference to better illustrate how SoftHub configuration objects are processed internally.

The Hub Main Component – This component is responsible for bringing up the overall system and initializing all other subcomponents. It will first read the users configuration file and instantiate all the user defined rule engine objects. It will then start each of the system subcomponents. Lastly, it starts the rule engine to carry out the user's ruleset. It then goes into a mode where it monitors for any outside communications (if enabled) as well as serving to monitor overall system health. Similarly, at shutdown it manages the orderly shutting down of each subcomponent.

The Rule Engine Subcomponent – This component continuously carries out the execution of the user defined ruleset. In general it waits on an internal event queue and as events are detected it will orchestrate the execution of any rules the user has associated with the specific event.

The ANT Comms Manager Subcomponent – This component manages all ANT sensor communications. It is responsible for initializing ANT services as well as creating and maintaining each individual ANT sensor connection. As packets are received this subcomponent will decode the packet and make the resulting packet data available to the rest of the system. Lastly a watchdog is implemented to monitor for, and reconnect, any dropped connections.

The BLE Comms Manager Subcomponent – This component manages all BLE sensor communications. It is responsible for initializing BLE services as well as creating and maintaining each individual BLE sensor connection. As packets are received this subcomponent will decode the packet and make the resulting packet data available to the rest of the system. Lastly a watchdog is implemented to monitor for, and reconnect, any dropped connections.

The GPIO Data Manager Subcomponent – This component is responsible for initializing the GPIO subsystem as well as configuring any user defined GPIO's as either input or output pins. It will then wait for input GPIO transition events on any enabled input pins and report them as events to the rule engine. Similarly, it carries out any requests from other SoftHub subcomponents to set or clear any user enabled output GPIO pins.

The MQTT Comms Manager Subcomponent – This subcomponent provides the MQTT client communications facilities within the hub. This is a full featured MQTT client allowing for user control of all standard MQTT protocol parameters. Additionally, the MQTT client code allows the user to perform last minute transformations on standard JSON packet data. This allows the SoftHub to publish data directly to most third party MQTT compliant cloud dashboards for visualization and further analysis. Additionally, the MQTT client includes a flexible interface for mapping incoming MQTT data to user defined Event objects. Using this facility, users can control their SoftHub remotely via either browser based dashboards or mobile applications.

The Ethernet Comms Manager Subcomponent – This subcomponent provides both client and server communications facilities within the hub. As a client it creates and maintains TLS encrypted client connections to any external servers running the OmnIoT Remote Packet Capture Application. Packet

data may be queued for each connection by any of the other SoftHub subcomponents. If the connection is available the data is sent immediately, however if communications are disrupted this component will optionally cache the data locally and resend once communications have been reestablished. As a server, this subcomponent maintains external client sessions for remote control of the SoftHub application. Lastly a watchdog is implemented to monitor for and reconnect any dropped client or server connections.

The Email Comms Manager Subcomponent – This subcomponent handles the sending of email or SMS messages to one or more recipients. As messages are queued from other SoftHub subcomponents, this module will attempt to successfully send them via the user's configured SMTP server. As with Ethernet connections, failed messages will be cached and retried until successfully sent.

The SysCommands Manager Subcomponent – This subcomponent handles the asynchronous queuing and execution of system commands. Using the ExecSysCommand Action Object, users can queue or execute immediately any system command or script.

The Logfile Data Manager Subcomponent – This subcomponent handles the logging of any data to local hub storage (including the SoftHub system log). It will create any user defined logfiles and manages the writing of packet or report data as received from any of the other SoftHub subcomponents.

The Timer Services Manager Subcomponent – This subcomponent is responsible for implementing any timer objects the user may define. As timer requests queued by external SoftHub subcomponents expire, this subcomponent is responsible for generating the associated system event objects to be consumed by the SoftHub Rule Engine subcomponent.

2.3 The SoftHub Rule Engine

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 controls 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 ruleset objects - Sensor Stream objects, User Report objects, Event objects, State objects, Action objects, and Rule objects. In order to effectively configure the SoftHub's rule engine it is vital to understand four basic concepts:

(1) Any SoftHub subcomponent may detect or generate events. When a specific event occurs an *Event Object* may 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 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 bound *State Objects* the user may have defined.

(4) If the optional conditional statement resolves to a Boolean “true” then one or more *Action Objects* associated with the rule will be executed (again in the order they are defined).

Understanding the above four tenants is central 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 summary for each object type as well as some specific example objects -

Event Objects – 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. Examples of some event objects include the *AppStarting* Event object, the *AppStopping* Event object, the *SensorConnect* Event object, and the *NewSensorPacket* Event object.

Action Objects – Action objects describe the individual actions to be carried when a Rule object’s conditional statement has been evaluated as “true”. Examples of some Action objects include the *ConnectSensor* Action object, *StartOneShotTimer* Action object, *SetGpioPinLow* Action object, and the *SendReport* Action object.

State Objects – State objects allow the user to associate conditional statements with the execution of a Rule object. Each State object will evaluate some aspect of the current system state as being either “true” or “false”. These objects in turn can be bound together via logical AND, OR, and NOT statements to create more complex logic. Examples of some state objects include the *SensorConnected* State object, *SensorDataValue* State object, *CounterValue* State object, and *FlagValue* State object.

Rule Objects – Rule objects themselves actually are mode up solely of references to other objects. A rule object will reference a single triggering Event object, an optional set of State objects, and 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. Each Action object in the list will carry out its one specific task when the conditional state logic has been resolved to “true”. An example Rule object could have a logic conditional that checked if “a specific sensor value was above a certain threshold AND it was after 5PM” then perform a list of action objects that might “enable a GPIO and send an SMS message to one or more recipients”.

Sensor Stream Objects – These objects define a sensor that you want the SoftHub to connect to. Note that unlike other object types, Sensor Stream objects don’t “do” anything on their own but rather are used as references in other object types. Examples of a Sensor Stream object would be a reference to a *TI SensorTag* BLE sensor, along with an optional MAC address if the user wanted to connect to only a specific physical sensor (as opposed to a specific sensor *type*).

Report Objects – Report objects allow the user to define their own custom tailored data packets, in either binary, JSON, or XML formats, from one or more sources. These packets in turn can be forwarded to a remote server or logged to internal storage on the hub device itself. Examples of data that can be included in these custom packets would be momentary sensor data values, GPIO values, cached sensor data, averaged sensor data, etc.. Report packets with no contents can be used to indicate event detection by correlating events to report ID’s (e.g. report id “1” could indicate a sensor is connected where event “0” could indicate a sensor has disconnected).

The above provides an overview of the various object types currently supported by the SoftHub's Rule Engine. In the addendum "Rule Engine Object Types" the full list of objects is briefly described, however a more in depth discussion of these objects and how the can be used can be found in a separate document titled "Configuring the OmnIoT SoftHub Rule Engine".

3. The SoftHub Configuration Utility

This section provides a brief overview of the SoftHub Configuration Utility application. A more detailed description of all the configuration options and rule engine objects available in the configuration utility can be found in the separate document titled *Configuring the OmnIoT SoftHub Rule Engine*.

The SoftHub Configuration Utility is a standalone application that runs on Windows PCs. This application serves two primary purposes. First it allows the user to set the global SoftHub System Options that will affect overall operation of the SoftHub. Examples of system options would be “Auto Connect Sensors” or “Allow Remote Login”. Second, the SoftHub Configuration Utility facilitates the defining of the rule engine objects and the rules themselves. Configuration files can be created from new, saved, and reedited as required.

The screenshot below shows the SoftHub Configuration Utility when creating a new configuration utility from scratch:

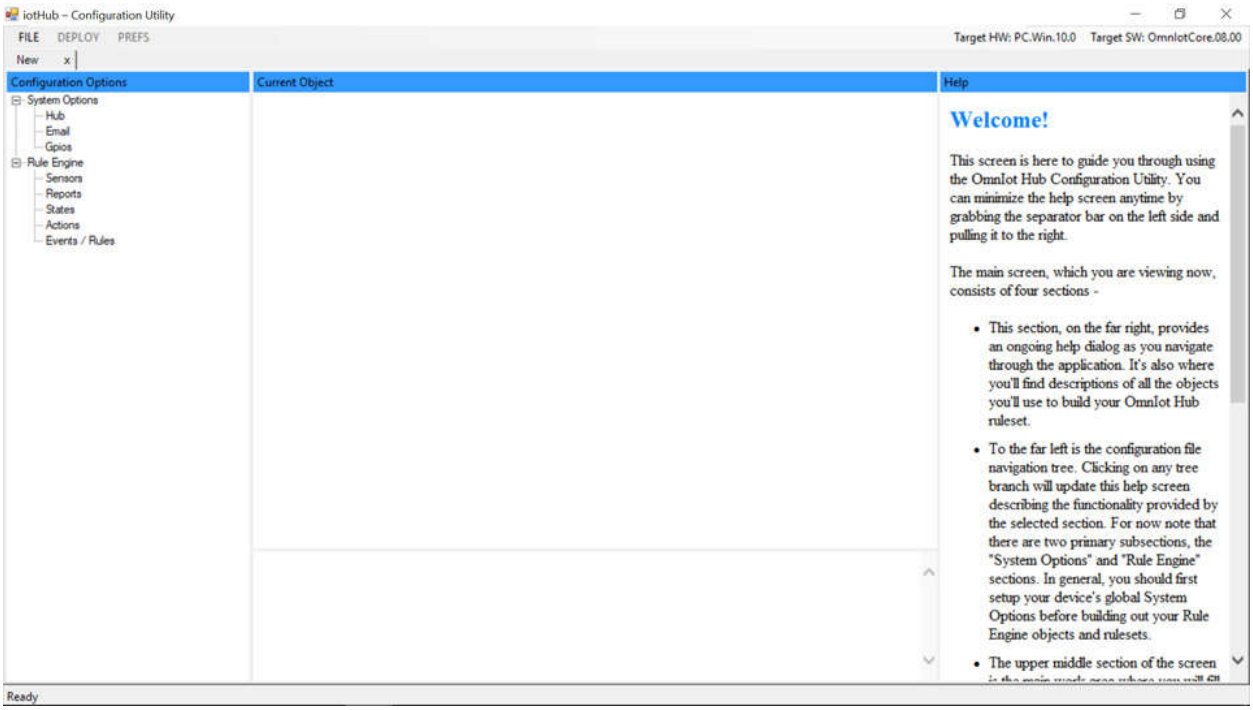


Figure 4: The SoftHub Configuration Utility Main Screen

The main screen is separated into four sections. First and foremost, on the left side of the screen is the *Configuration Options* panel displaying the configuration object navigation tree. To the far right is a collapsible context sensitive *Help* screen panel. The main panel (upper center), titled *Current Object*, is where any currently selected object in the navigation tree would be defined or edited. And at the bottom center of the screen is an *Object Information* panel providing additional object information, in particular an object cross reference is displayed.

3.1 The Configuration Options Panel

This panel allows you to navigate the objects that constitute the Rule Engine Configuration File. There are two major sections, (1) the *System Options* section and, (2) the *Rule Engine* section.

The System Options section is where you define global system settings for your hub device. This section is further broken down into three subsections. The first, the *Hub* section, allows you to configure general hub settings. Examples include giving your hub device a unique identifier or allowing or disallowing remote login to your hub. The second section, the *Email* section, allows the user to configure the hub to be able to send emails or SMS messages directly from the hub device via an SMTP server. The third section, the GPIOs section, allows the user to define any input or output GPIO pins the user would like to enable. Left clicking on any of the three entries in the navigation tree will populate the Object Editor panel with all the options for that section. There the user can make and save any desired changes.

The Rule Engine section is where the user defines all the objects to be incorporated into their ruleset. Left clicking on any of the five subsections will expand the tree to expose any previously defined objects of the specific type. Right clicking will open a dialog pop-up where the user can choose to create a new object from a list of specific objects types available for the chosen subtype.

3.2 The Current Object Panel

As System Options or Rule Engine objects are selected or created via the object tree navigation panel, the Current Object Panel will be populated with all the values that can be edited for the particular chosen system option or rule engine object. Once changes are made to the users' satisfaction clicking on the *Save* button will commit any changes made. In the event the user has entered an invalid value a red "X" will appear next to the value in error. To resolve invalid value errors, reference the context sensitive Help panel where a description and valid value range information is provided for each option.

3.3 The Object Help Panel

To the right of the Object Editor Panel is the context sensitive object Help Panel. As the user navigates through the tree displayed in the Configuration Options panel, this screen is constantly updated to present help information for the currently selected item. Information contained in these screens include (1) what mouse control submenus are available, (2) an overview of each value presented in the Current Object editor panel, and (3) range limitations on what values can be entered for a specific parameter. Note that this screen can be always be minimized to expand the Current Object screen size if desired.

3.4 The Object Information Panel

When creating a SoftHub ruleset objects will often reference other objects. When deleting or otherwise altering an object it may be necessary to make changes to any externally referenced objects as well. The Object Cross Reference Panel lets the user easily determine what other objects the current object references or is referenced by.

4. The Remote Packet Capture Application

The Remote Packet Capture Application runs as a System Service on computers running Microsoft Windows (a Linux version is planned in the future). This application provides a few basic functions. Primarily, it monitors incoming connections on a user defined port and receives packets from one or more remote hub devices running the SoftHub application. As packets are received they will be made available to any user application in both a raw binary and decoded JSON or XML formats. Users provide their own application to receive this data by creating a DLL named "PacketReceiver.dll" and placing it in the same directory as the Service executable. This DLL should export one entry point as –

```
void PacketRecvFunc(char* pDecodedPacket, char* pRawPacket, unsigned int rawPacketSize);
```

If the DLL is found when the service starts up, every new arriving packet will be decoded and passed to this DLL. For each call *pDecodedPacket* will point to the null terminated JSON or XML decoded packet string, *pRawPacket* will point to the raw binary packet used to generate the decoded packet, and *rawPacketSize* will indicate the size of the binary data pointed to by pRawPacket. A project template is available to aid the user in creating the PacketReceiver DLL. An example of an ANT Tempe sensor packet decoded in XML is provided below for reference:

```
<?xml version="1.0" encoding="UTF-8"?>
<OmnIotHubPacket>
   <OriginDeviceId>555</OriginDeviceId>
   <PacketGeneratedTimestamp>05/22/2018 03:51:23 232:184 UTC</PacketGeneratedTimestamp>
   <PacketReceivedTimestamp>05/22/2018 03:51:23 452:509 UTC</PacketReceivedTimestamp>
   <SequenceNumber>6</SequenceNumber>
   <PacketType>Raw ANT</PacketType>
   <PacketContents>
      <SensorType>Ant Environment(25)</SensorType>
      <AntChannel>0</AntChannel>
      <AntPageNum>1</AntPageNum>
      <PacketDataField>
         <FieldName>24hr Low Temp Celsius</FieldName>
         <FieldValue>20.6</FieldValue>
         <FieldValueHex>CDCCA441</FieldValueHex>
      </PacketDataField>
      <PacketDataField>
         <FieldName>24hr High Temp Celsius</FieldName>
         <FieldValue>23.200001</FieldValue>
         <FieldValueHex>9A99B941</FieldValueHex>
      </PacketDataField>
      <PacketDataField>
         <FieldName>Current Temp Celsius</FieldName>
         <FieldValue>22.15</FieldValue>
         <FieldValueHex>3333B141</FieldValueHex>
      </PacketDataField>
      <PacketDataField>
         <FieldName>Measurement Count</FieldName>
         <FieldValue>122</FieldValue>
      </PacketDataField>
   </PacketContents>
</OmnIotHubPacket>
```

A similar companion utility, the "OmnIoTLogDecoder" line command, is also provided as part of the Packet Capture Application installation materials. This utility can be used to decode and further process any binary log files created directly on the SoftHub itself via the “LogData” action object. As with the Packet Capture application, the Log File Decode Utility ultimately passes the decoded packets to a user created DLL for further processing. Note that the calling convention is the same for both applications so the same DLL can be shared between the two applications if desired.

To use the Log File Decode utility, copy or move your log file from the hub device to the windows computer where you will run the utility. The utility itself accepts either one or two parameters. To process a single log file, enter that log file’s fully qualified filename as the sole parameter when executing the decode utility. For instance, to decode a file stored at "D:\SoftHub Logs\SensorData.log" from a Windows Command prompt you would enter -

> OmnIoTLogDecoder "D:\SoftHub Logs\SensorData.log"

To process more than one file at a time create a separate file containing a list of files to process, each fully qualified and on a separate line. Then call the utility using the "-l" parameter to indicate you are passing a list file. For example -

> OmnIoTLogDecoder -l "D:\SoftHub Logs\LogFileList.txt"

This will read LogFileList.txt and then consecutively process each log file listed within.

Note, as with the Packet Capture Application the utility passes each decoded packet to a user supplied DLL which must reside in the same directory as the utility itself. In the installation materials there are example dll's provided as well as example visual studio projects to help you build your own to meet your requirements.

5. MQTT Connections

In addition to forwarding data to an OmnIoT Remote Packet Capture server, the SoftHub provides the ability to publish your data to any standard MQTT broker in a variety of formats. The SoftHub provides access to all standard MQTT connection settings, including Last Will and SSL/TLS parameters, to ensure compatibility with most, if not all, third party cloud based solutions. Additionally, outgoing data packets can be easily reformatted on the fly to meet most vendor's requirements. As well, incoming control packets can be evaluated and mapped to trigger Events and Rules in your SoftHub ruleset from remote dashboards and mobile apps. To fully take advantage of the SoftHub's MQTT capabilities, users should familiarize themselves with the associated objects as well as the MQTT "Extended Options Files" available to configure MQTT connections. These are all covered in detail in the OmnIoT SoftHub Configuration Guide but a brief summary of the various MQTT related components is provided below:

- **The "PublishData" and "PublishReport" Action objects** - These action objects allow the user to publish raw sensor data or user defined custom report packets (respectively) to a user specified MQTT broker using a user specified topic. These objects allow the user to configure the basic connection parameters as well as specify the format the data should be published in (JSON, XML, or raw Binary).

- **The "UserDefinedEvent" Event** object - These event objects can be used to trigger the evaluation of rules based on incoming control packets received from remote MQTT brokers or dashboards. By configuring which received packets to respond to, users can control what functionality they choose to expose, from none or little to virtually any action the SoftHub can perform (including executing any system command or script).

- **MQTT "Extended Options" (.mcfg) files** - these files allow the user to control all extended parameters available when creating an MQTT connection. While the most common connection parameters can be specified when creating the PublishData and PublishReport Action objects, some parameters allowing for finer control may still be set using these extended configuration files (e.g. "Last Will" and extended "SSL/TLS" parameters). In addition to the extended connection parameters these files allow the user to control one of the most important features of the SoftHub's MQTT functionality, the ability to reformat outgoing packets as required for the many third party MQTT brokers as well as the ability to map set incoming MQTT control messages to User Defined Events to be queued.

- **"JSON Filter Format" (.jff) files** - These files specify any specific filtering and formatting required to be performed on outgoing packets. As packets are generated by the SoftHub, JSON Filter Format file information allows the user to do last minute transformation on data packets. This is important where a target MQTT broker does not support the ingesting of generic standard JSON data.

As mentioned above, all of these components are described in detail in the OmnIoT SoftHub Configuration Guide as well as the SoftHub Configuration Utility's help facility and the template .mcfg and .jff files themselves (found in the default rule engine configuration file directory).

6. Tying It All Together

Detailed installation instructions for each of the OmnIoT SoftHub Platform components are included with the platform installation package available for download at the OmnIoT website (omniot.com). In general however, the high-level steps to deploy an IoT application built on the SoftHub platform include the following:

(1) Acquire the sensor and hub hardware appropriate for your application
(2) Download the SoftHub software installation materials from the link(s) provided at the OmnIoT website (www.omniot.com).
(3) Load the SoftHub application ".img" file onto your hub device(s), install the SoftHub Configuration Utility application on a Windows PC, and install the optional Remote Packet Capture Application on one or more remote packet capture servers if required.
(4) Run the SoftHub Configuration Utility and create the SoftHub configuration file(s) per the "Configuring the OmnIoT SoftHub Rule Engine" manual.
(5) Distribute the configuration files as described in the installation instructions., restart the SoftHub Daemon on each updated hub device.

Addendum A: Rule Engine Object Types

In this section we describe briefly each of the different objects currently available for use with the SoftHub's rule engine. Note that this list will grow significantly as new subcomponents are incorporated and additional functionality added.

A.1 Sensor Stream Objects

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.

A.2 Report Objects

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.

A.3 Event Objects

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.

A.4 Action Objects

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.

A.5 State Objects

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.

A.6 Rule Objects

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).