

Intel® RSP SW Toolkit - Gateway

Installation & User's Guide

Document Number: 338443-002

Document Revision: 2019.3.22

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting: <http://www.intel.com/design/literature.htm>

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at <http://www.intel.com/> or from the OEM or retailer.

No computer system can be absolutely secure.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2019, Intel Corporation. All rights reserved.

Revision History

Version	Revision	Description
	2018.11.14	Initial draft.
	2018.12.11	Updated screenshots.
	2018.12.13	Branding updates.
338443-001	2018.12.20	Fixed missing reference for behavior. Removed stale TODO comment
338443-002	2019.3.22	Updated contents for clusters, web admin and software update capability Addressed path discrepancies in build/deploy instructions.

Table of Contents

1	INTRODUCTION	7
1.1	TERMINOLOGY	7
1.2	REFERENCE DOCUMENTS	7
2	SENSOR PRODUCT DESCRIPTION	8
3	SYSTEM DESCRIPTION	9
3.1	DATA FLOW	9
3.2	APPLICATIONS	10
3.3	GATEWAY	10
3.4	INTEL® RFID SENSOR PLATFORM (INTEL® RSP)	10
3.5	JSON RPC	11
3.5.1	Request Object	11
3.5.2	Notification Object	11
3.5.3	Response Object	12
3.5.4	Error Codes	12
3.6	MQTT	13
3.7	REST	13
3.8	SYSTEM REQUIREMENTS	14
3.8.1	Hardware Requirements	14
3.8.2	Software Requirements	14
3.8.3	Network Requirements	14
4	CONCEPT OF OPERATION	15
4.1	CONFIGURATION: SENSOR FACILITY	15
4.2	CONFIGURATION: SENSOR PERSONALITY	15
4.2.1	EXIT Personality	15
4.2.2	Point of Sale (POS) Personality	16
4.2.3	FITTING_ROOM Personality	16
4.3	CONFIGURATION: RFID BEHAVIORS	16
4.3.1	WITH_TID	17
4.3.2	PORTS_#	17
4.3.3	Example Behavior	17
4.4	CONFIGURATION: CLUSTERS	18
4.5	SCHEDULING TAG READS	19
4.5.1	INACTIVE	19
4.5.2	ALL_ON	19
4.5.3	ALL_SEQUENCED	19
4.5.4	FROM_CONFIG	19

4.6	PROCESSING TAG READS	21
4.6.1	Tag Location.....	21
4.6.2	Tag States.....	23
4.6.3	Tag Events.....	23
4.6.4	Tag Processing Algorithm.....	23
4.7	UPSTREAM EVENTS.....	25
4.7.1	Inventory Event.....	25
4.7.2	Gateway Device Alert.....	26
5	<u>BUILD AND DEPLOY</u>	<u>27</u>
5.1	LINUX.....	27
5.1.1	Install Dependencies.....	27
5.1.2	Clone the Project.....	27
5.1.3	Build and Deploy	27
5.2	WINDOWS® 10.....	28
5.2.1	Install Dependencies.....	28
5.2.2	Edit the Path Environment Variable	28
5.2.3	Download the Project.....	32
5.2.4	Build and Deploy	33
5.3	PROJECT RUNTIME FOLDERS.....	34
5.4	CONFIGURE	35
5.4.1	Windows® 10.....	37
5.5	CERTIFICATE GENERATION.....	38
5.5.1	Linux	38
5.5.2	Windows® 10.....	38
5.6	RUN.....	39
5.6.1	Linux	39
5.6.2	Windows® 10.....	39
5.7	SENSOR CONNECTION SEQUENCE	40
6	<u>SYSTEM PROVISIONING CLUSTER TOKENS</u>	<u>41</u>
7	<u>WEB ADMINISTRATION.....</u>	<u>44</u>
8	<u>COMMAND LINE INTERFACE (CLI)</u>	<u>45</u>
8.1	EXIT - QUIT	45
8.2	SCHEDULER	45
8.3	SYSTEM.....	46
8.4	UPSTREAM.....	46
8.5	DOWNSTREAM	46
8.6	LOG.....	46
8.7	SENSOR.....	46

8.8 INVENTORY	48
8.9 VERSION	49
8.10 CLUSTERS	49
<u>9 SENSOR SOFTWARE UPDATE.....</u>	<u>50</u>

1 Introduction

This document is a guide to the installation, setup and use of the Gateway Application Reference Design, which is one component of the Intel® RSP SW Toolkit. The features and functionality included are intended to showcase the capabilities of the Intel® RFID Sensor Platform (Intel® RSP) by demonstrating the use of the API to collect and process RFID tag data. THIS SOFTWARE IS NOT INTENDED TO BE A COMPLETE END-TO-END INVENTORY MANAGEMENT SOLUTION.

1.1 Terminology

Term	Description
RSP	RFID Sensor Platform
NFC	Near Field Communications
Deep Scan Behavior	A particular set of RFID protocol parameters that is intended to read as many tags as possible but the read rate is lower.
Mobility Behavior	A particular set of RFID protocol parameters that is intended to read tags in motion as quickly as possible.

1.2 Reference Documents

Document	Document No./Location
RRS-Hx000_User_Guide	338088-001
RRS-Hx000_Message_API	338178-001
Intel® RSP SW Toolkit - Sensor NFC App User Guide	338454-001

2 Sensor Product Description

The RSP-9000, RSP-H1000, RSP-H3000 and RSP-H4000 are members of the Intel® RFID Sensor Platform (Intel® RSP) family of devices. These devices have capabilities for several on-board sensors including an EPC Gen 2 UHF RFID Interrogator (reader). These Sensors are designed to work stand-alone, or in a network of other “Smart Sensors” as part of an Internet-of-Things (IoT) system where computing power is pushed out to the edge devices.



Figure 1: Intel® RFID Sensor Platform (Intel® RSP)

The Gateway Reference is an application that demonstrates the use of the API to collect and process RFID tag data and acts as the IOT Gateway between one or more RSP's and an Inventory Management or Asset Tracking application. The Gateway Reference Design creates the credentials used by the Sensor NFC Application (another component of the Intel® RSP SW Toolkit).

3 System Description

A complete end-to-end Inventory Management solution might look something like the figure below. One or more Sensors sending raw data to a Gateway that processes the raw data and creates meaningful events, which are sent to an Inventory Management Application that can exist locally or in the Cloud.

3.1 Data Flow

From a data flow perspective, the Sensor interrogates the RFID tag population within its field of view and passes information regarding the tags as well as information from other various on-board sensors to the Gateway. The Gateway aggregates this Sensor data and generates inventory events, alerts, and system status notifications available for consumption on an upstream channel to applications running in a customer's cloud infrastructure. The figure below illustrates the flow of data and control within the system.

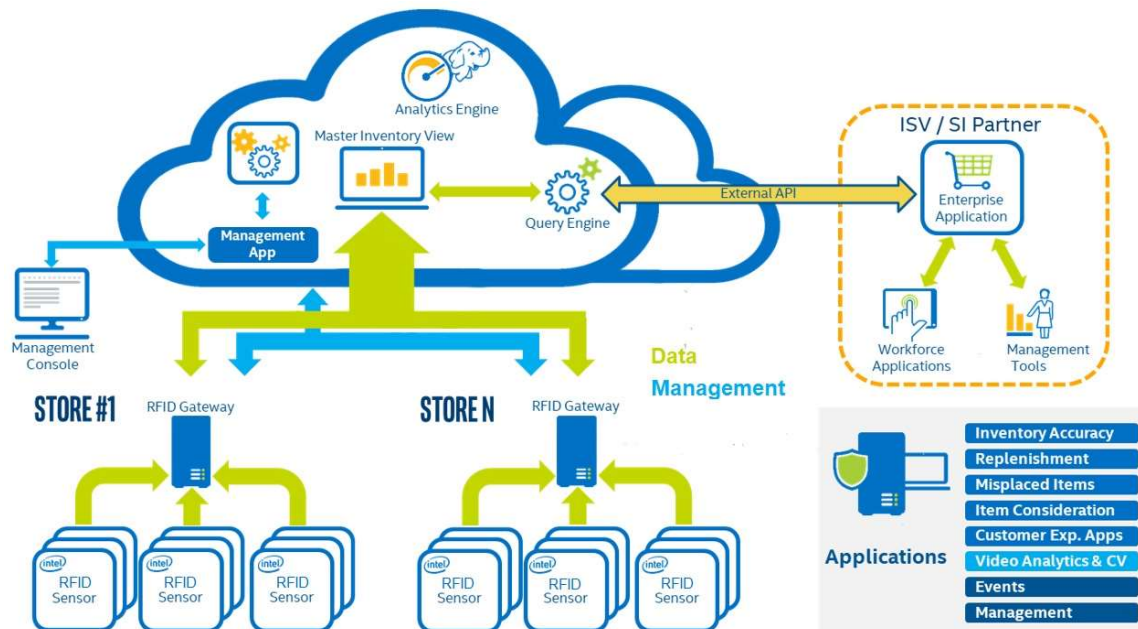


Figure 2: Example System Data Flow

3.2 Applications

Customers may utilize their own infrastructure for applications that ingest the events from the Gateway allowing them to determine item identification, location, movement and status. CUSTOMERS ARE SOLELY RESPONSIBLE FOR THEIR OWN CLOUD INFRASTRUCTURE AND APPLICATIONS, INCLUDING FOR IDENTIFYING AND IMPLEMENTING THE APPROPRIATE LEVEL OF SECURITY FOR THE DATA COLLECTED VIA THE DEVICES.

3.3 Gateway

The Gateway performs Sensor control, management, data aggregation, data processing, and event management as well as event generation for upstream consumption all over a secure data channel. It also supports configuration and management from a local interface. CUSTOMERS ARE SOLELY RESPONSIBLE FOR CONFIGURING THEIR MQTT BROKER FOR THE APPROPRIATE LEVEL OF SECURITY FOR THE DATA COLLECTED VIA THE DEVICES.

3.4 Intel® RFID Sensor Platform (Intel® RSP)

The RSP devices provide the ability to remotely and securely command, control, status, and data collection via Ethernet. Data from RFID tag reads as well as data from other on-board sensors is published to an MQTT broker. The data API is based on JSON RPC requests, responses and notifications. JSON-RPC is a text based, stateless, lightweight remote procedure call (RPC) protocol.

3.5 JSON RPC

A secure remote capability for command, control, status and data collection exists via JSON Remote Procedure Call (RPC) over an encrypted MQTT channel. CUSTOMERS ARE SOLELY RESPONSIBLE FOR CONFIGURING THEIR MQTT BROKER FOR THE APPROPRIATE LEVEL OF SECURITY FOR THE DATA COLLECTED VIA THE DEVICES.

The Gateway Command set follows the JSON RPC 2.0 specification. JSON-RPC is a stateless, lightweight protocol that is transport agnostic.

3.5.1 Request Object

The Request object has the following members:

- **jsonrpc**
 - A String specifying the version of the JSON-RPC protocol.
- **method**
 - A String containing the name of the method to be invoked.
- **params**
 - A Structured value that holds the parameter values to be used during the invocation of the method.
 - This member may be omitted.
- **id**
 - An identifier containing a String or Number value (if included).
 - This member is used to correlate the context between requests and responses.

3.5.2 Notification Object

A Notification is a Request object without an "id" member. A Request object that is a Notification signifies that a corresponding Response object is not expected.

3.5.3 Response Object

The Response is expressed as a single JSON Object, with the following members:

- **jsonrpc**
 - A String specifying the version of the JSON-RPC protocol.
- **result**
 - The presence of this member indicates successful execution of the corresponding method.
 - This member is not present when the execution of the method resulted in an error.
- **error**
 - The presence of this member indicates unsuccessful execution of the corresponding method.
 - This member is not present when the execution of the method was successful.
 - When present, the error Object contains the following members:
 - **code**
 - An integer that indicates the error type that occurred.
 - **message**
 - A String providing a short description of the error.
 - **data**
 - A Primitive or Structured value that contains additional information about the error (optional).
 - See table below for supported error codes.
- **id**
 - This member is always present on a response and contains the same value as the id member in the corresponding Request Object.
 - This member is not present on indications.

3.5.4 Error Codes

The RSP provides one of the following error codes when an error occurs.

Table 1 JSON RPC Error Code Fields

Code	Message	Meaning
-32001	Wrong State	Cannot be executed in the current state
-32002	Function not supported	The requested functionality is not supported
-32100	No facility assigned	The RSP has no Facility ID assigned yet
-32601	Method not found	The method does not exist
-32602	Invalid Parameter	Out of range or invalid format
-32603	Internal Error	RSP application error
-32700	Parse error	Invalid JSON Object

3.6 MQTT

The Sensor supports the MQTT machine-to-machine “Internet of Things” connectivity protocol. By subscribing and publishing to a set of “topics”, the Sensors coordinate with the Gateway. Sensors support TLS encrypted MQTT connections by using the server certificate and MQTT credentials assigned by the Gateway.

NOTE: Certificate management is important when using TLS encryption on the MQTT broker. If the root certificate used for the MQTT broker is not publicly trusted, it must be the same root certificate used by the Gateway’s REST endpoints.

3.7 REST

The Sensor obtains necessary configuration data via REST endpoints. These endpoints are provided to the Sensor by the Gateway as part of the “zeroconf” discovery process described later in this document. A simple data architecture diagram is shown below. The Gateway endpoints are secured using the root certificate and the server certificate.

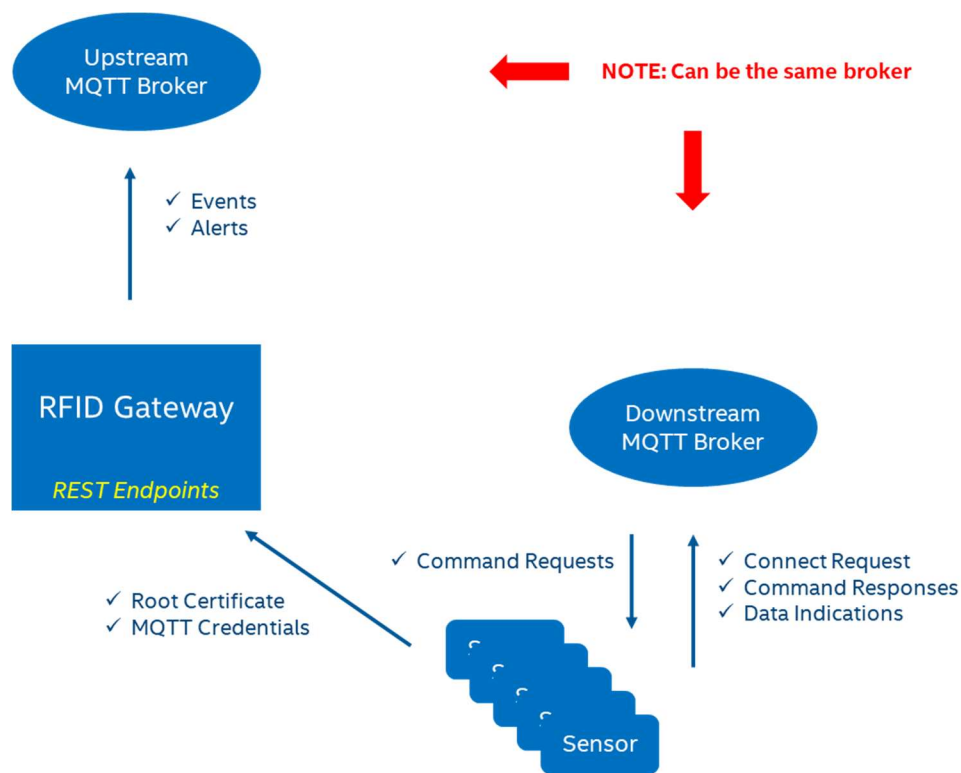


Figure 3: Simple Data Flow Architecture

3.8 System Requirements

To ensure proper functionality of the Gateway, the following requirements should be observed for both the hardware platform that the application is running on and the network to which it's attached.

3.8.1 Hardware Requirements

To support the complex algorithms necessary for real-time tag location and dynamic Sensor management, it is recommended the Gateway should be installed on a machine with the following minimum configuration.

- ✓ CPU: Intel® Core® i5, 8MB Cache, Intel® vPro, Intel® AMT Technology
- ✓ RAM: 16 GB minimum
- ✓ SDD: 80 GB minimum

3.8.2 Software Requirements

The Gateway is a Java application. As such, it can run on any OS that supports a Java Runtime Environment version 8 or greater. The development of the Gateway software has been done on an Ubuntu Linux platform and there are several bash scripts supporting installation and configuration, so to enable out of the box (repository) operation, a Linux OS is recommended.

3.8.3 Network Requirements

To facilitate the “zeroconf” discovery process between Gateway and Sensors, it is recommended that all the devices be on the same LAN segment. The table below lists the protocols and typical ports used for the network communications between the Gateway and Sensors.

Table 2 Local Network Protocols and Ports

Port	Protocol	Purpose
5353	mDNS	Service discovery of Gateway and Sensors
1883, 9883	tcp, ssl	MQTT message transport
80, 443, 8080, 8443	tcp	http, https for software management
22, 62939, 5222	ssh	Gateway and Sensor shell connectivity for trouble shooting and log viewing
123	ntp	Time synchronization across Sensors

4 Concept of Operation

The following functionalities are supported by this reference design.

- Configuration and management of a population of Sensors which includes actively coordinating the reading of tags by each Sensor using scheduling and behaviors.
- Processing tag data produced by each Sensor.
- Determining an approximate tag location via algorithms that consider the signal strength, the behavior used to generate the tag read, and a mobility profile that enables customization of the algorithm.
- Generating various tag events (arrived, moved, departed, returned) depending on changes of a tag's location. The generation of these events are influenced by Sensor facility and personality configurations. These events are published to an upstream MQTT broker.
- Providing real time visibility into the performance of the system by presenting aggregated statistics for Sensors and tags.

There are two Communication channels identified for the gateway

- *Downstream* - between Sensor(s) and gateway
- *Upstream* - between gateway and any 3rd party applications consuming data from the Upstream MQTT broker.

4.1 Configuration: Sensor Facility

Facilities are used to create zones that govern tag event generation i.e. a tag will arrive in a facility, a tag will depart one facility and arrive in another facility. A tag will move within a facility. Each Sensor must be assigned to one, and only one facility and the tags read by that Sensor are associated with that facility. The Gateway can support multiple facilities. It is acceptable to assign all Sensors managed by a Gateway to the same facility.

Facilities can be assigned to Sensors either via the command line interface or a schedule configuration.

4.2 Configuration: Sensor Personality

Personalities are used to optionally identify special functionality for a Sensor. Tag reads from these Sensors have unique meaning and are handled differently by the tag processing algorithms.

4.2.1 EXIT Personality

As the name implies, an EXIT personality indicates that the Sensor is located at or near the exit of a facility. Tags that are from these are highly likely to be on their way out. A

DEPARTED event is generated by the Gateway when RFID tags are last seen by an EXIT Sensor and are not seen again for a configurable amount of time.

4.2.2 Point of Sale (POS) Personality

A POS (Point-of-Sale) personality is used when tag reads coming from these Sensors will immediately generate a departed event. The use-case for configuring a Sensor with a POS Personality might be either a store checkout or exit portal. The time delay for when a tag is allowed to depart and when it is allowed to re-enter the inventory is configurable. See the INVENTORY section of the gateway.cfg file.

4.2.3 FITTING_ROOM Personality

This personality is only useful as a reference. There is no special handling of tag reads associated with it.

4.3 Configuration: RFID Behaviors

RFID Behaviors, in general, contain ISO 18000-6C protocol specific parameters as well as other information regarding the RF behavior of the Sensor. See the Example Behavior section for details of the parameters. The gateway includes several default behaviors that are appropriate for tag read scenarios such as mobility which emphasizes reading tags that are in motion and deep scan which emphasizes long scan times and reading as many unique tags as possible.

The Gateway also supports adding custom behaviors by simply adding JSON files to the behaviors configuration directory. See the Deployment section for details of where to add new behavior configurations.

Shown below are included behaviors.

```
DefaultAllOn.json
DefaultAllSeq.json
DefaultDeepScan.json
DefaultExit.json
DefaultFittingRoom.json
DefaultMobility.json
DefaultPOS.json
```


Concept of Operation

Specific naming criteria is used to differentiate some of the default behaviors including influencing the behavior of the Sensor based on keywords included in the behavior id. NOTE !! The behavior id and the filename must match, excluding the .json extension.

4.3.1 WITH_TID

Any behavior that includes this token in the id will cause the sensor to read the TID memory bank of the RFID tags.

NOTE!! Reading TID requires additional RFID protocol activity and as such, tag read rates are much lower.

4.3.2 PORTS_#

Any behavior that includes this token in the id indicates the minimum number of antenna ports required. For example, the H1000 has four antenna ports available. The installation requirements are for three ports to be connected. The system should send alerts if one of the cables is loosened or damaged and unusable.

Create and use a behavior with PORTS_3 in the id. When the sensor sees this behavior, it will verify that there are at least 3 good antenna connections. If there are less ports available, the sensor sends an alert to the gateway.

4.3.3 Example Behavior

Here is an example of the contents of a behavior.

git repo: config/behaviors/DefaultAllOn.json

```
{
  "id": "DefaultAllOn",
  "operation_mode": "NonContinuous",
  "link_profile": 1,
  "power_level": 30.5,
  "selected_state": "Any",
  "session_flag": "S1",
  "target_state": "A",
  "q_algorithm": "Dynamic",
  "fixed_q_value": 10,
  "start_q_value": 7,
  "min_q_value": 3,
  "max_q_value": 15,
  "retry_count": 0,
  "threshold_multiplier": 2,
  "dwell_time": 5000,
  "inv_cycles": 0,
  "toggle_target_flag": true,
  "repeat_until_no_tags": false,
  "perform_select": false,
  "perform_post_match": false,
  "filter_duplicates": false,
  "toggle_mode": "OnInvCycle",
  "auto_repeat": false
}
```

4.4 Configuration: Clusters

Cluster Configurations are used to manage a population of Sensors. Several configuration topics are all combined into one text file which handles assigning facilities and personalities, grouping of Sensors to minimize interference, and selecting behaviors to apply for the Sensor to use when reading tags. The following concepts help to understand the configuration.

- A Sensor Group is a collection of one or more sensors that will be managed as a group. Sensors in the group will all read tags at the same time.
- A Cluster contains one or more Sensor Groups and has an associated Facility, Behavior, and optional Personality.

The following JSON is an example of a cluster configuration using sensor device ids:

git repo: src/test/resources/clusters/sample_sensor_cluster_config.json

```
{
  "id": "SampleSensorClusterConfig",
  "clusters": [
    {
      "id": "BackStockDefault",
      "facility_id": "BackStock",
      "behavior_id": "DefaultDeepScan",
      "sensor_groups": [
        ["RSP-000000"],
        ["RSP-000001"]
      ]
    }, {
      "id": "SalesFloorDefault",
      "facility_id": "SalesFloor",
      "behavior_id": "DefaultMobility",
      "sensor_groups": [
        ["RSP-000002", "RSP-000004", "RSP-000006"],
        ["RSP-000003", "RSP-000005", "RSP-000007"]
      ]
    }, {
      "id": "SalesFloorExit",
      "facility_id": "SalesFloor",
      "behavior_id": "DefaultExit",
      "personality": "EXIT",
      "sensor_groups": [
        ["RSP-150008"]
      ]
    }, {
      "id": "SalesFloorPOS",
      "facility_id": "SalesFloor",
      "behavior_id": "DefaultFittingPOS",
      "personality": "POS",
      "sensor_groups": [
        ["RSP-150009"]
      ]
    }, {
      "id": "SalesFloorFittingRoom",
      "facility_id": "SalesFloor",
      "behavior_id": "DefaultFittingRoom",
      "personality": "FITTING_ROOM",
      "sensor_groups": [
        ["RSP-150010"]
      ]
    }
  ]
}
```

4.5 Scheduling Tag Reads

Scheduling a population of Sensors tag reading activity is important because of the following:

- *Managing Interference* - When nearby Sensors are actively reading at the same time, the RF energy can cause interference. It is important to recognize and minimize this interference. This can be done by the location of the Sensors and also by grouping Sensors together based on non-interference and then scheduling these groups to actively read at the same time.
- *Managing RFID Read Parameters (Behaviors)* - Behaviors are used to describe a set of parameters that can be applied to a Sensor while it executes tag reads. As described below, a Sensor is sent a behavior configuration when commanded to start reading tags. See section 4.3.3 for more details of a behavior.

The schedule manager coordinates different modes of Sensor read activity including starting and stopping Sensors according to a configurable sequence. This manager is controlled via the command line interface and any changes applied from there are persistent across restarts of the gateway. The following describe the possible run states of the schedule manager.

4.5.1 INACTIVE

The scheduler is not actively commanding any Sensors to read tags.

4.5.2 ALL_ON

All Sensors will be reading constantly using the **default_all_on** behavior.

4.5.3 ALL_SEQUENCED

A single Sensor at a time will be reading using the **default_all_sequenced** behavior. This mode does a round robin sequence of every Sensor connected to the gateway.

4.5.4 FROM_CONFIG

The schedule manager will create a cluster runner for each of the clusters defined in the configuration. The Facility and (optional) Personality will be assigned to the Sensors in the cluster and then a control algorithm is used to iterate through the groups of Sensors. For each group, the behavior is applied and all Sensors are commanded to start reading tags. After all Sensors in the group have finished reading tags or timed out, as

determined by the Behavior parameters, the next group of Sensors is started. The groups are continuously looped until the schedule manager changes run state.

It is perfectly fine to have a cluster with just a single sensor group that contains a single sensor.

To manage RFID interference, Sensors need to be placed in groups such that interfering Sensors are not in the same group in the cluster.

4.6 Processing Tag Reads

Tag read data includes information such as the EPC, RSSI of the read and includes the antenna port that the Sensor used. These tag reads are ingested from the Downstream MQTT broker and processed as Inventory data using the EPC as a unique key. Sensor and tag read statistics are aggregated and updated and then location and event generation algorithms are executed on the updated tag data.

4.6.1 Tag Location

An RFID tag's location is associated with a particular Sensor's antenna port. A location is determined based on the "quality" of the tag reads (i.e. RSSI, read rate) averaged over time. For a tag's location to move from one Sensor/antenna to another Sensor/antenna, the reported new RSSI must be better than a calculated threshold that is added to the current location's RSSI. A "mobility profile" defines the parameters of the RSSI slope formula used to calculate this threshold.

If the tag is read with a Deep Scan behavior, the threshold value itself is used, otherwise the threshold value decays over time according to the parameters in the mobility profile. Mobility profiles are configurable by the user.

Parameters:

T - RSSI threshold (dBm) that must be exceeded for the tag to move from the previous sensor
M - Slope (dBm per millisecond) used to determine the weight applied to older RSSI values
A - Hold-off time (milliseconds) to shift the slope such that when time == A, the weight = T

Algorithm:

```
// find b such that at 60 seconds, y = 3.0
// b = y - (m*x)
B = T - (M * A);
// weight used in the RSSI comparison
w = (M * (System.currentTimeMillis() - _lastReadMillis)) + B;
// weight cannot be more than the absolute threshold
if (w > T) { w = T; }
```

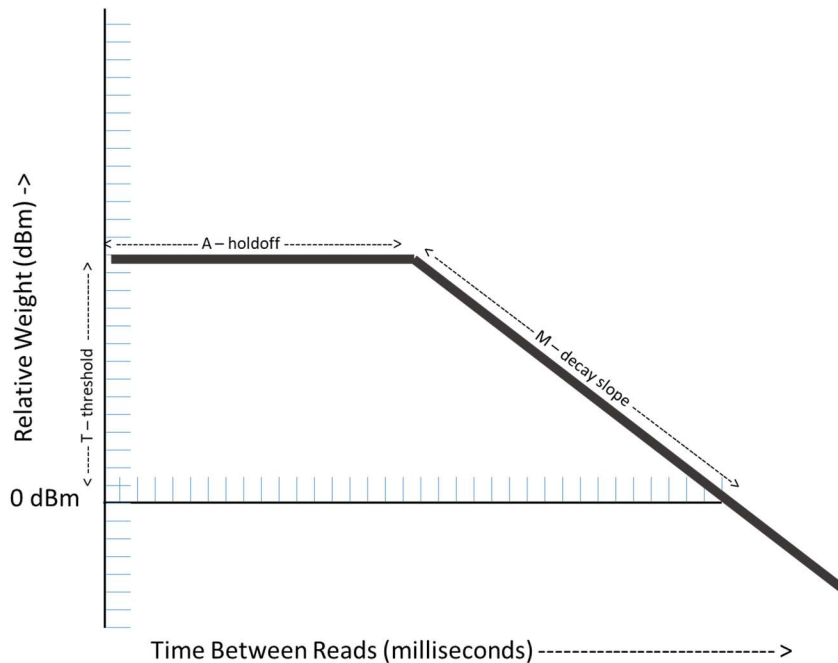


Figure 4 Mobility Profile

These profiles are JSON files and customized profiles can be used by adding the file to the "<DEPLOY_DIR>/config/mobility" directory. The following profiles exist by default with the RSP-Gateway-Demo software.

config/mobility/asset_tracking_default.json

```
{
  "id" : "asset_tracking_default",
  "a" : 0.0,
  "m" : -8.0E-3,
  "t" : 6.0
}
```

config/mobility/retail_garment_default.json

```
{
  "id" : "retail_garment_default",
  "a" : 60000.0,
  "m" : -5.0E-4,
  "t" : 6.0
}
```

4.6.2 Tag States

There are five possible states for a tag.

- *unknown* - This is an initial state that only exists momentarily upon tag data creation. It should actually never be visible from any user perspective.
- *present* - The tag has been read and is currently in the inventory.
- *exiting* - The tag has been read by a Sensor with an Exit personality and potentially will be departing.
- *departed exit* - An exiting tag that has not been seen for some amount of time is assumed to no longer be in the inventory.
- *departed pos* - A tag that is read by a Sensor with POS personality will immediately enter this state.

4.6.3 Tag Events

The transition of tags between states will generate events that are sent Upstream. The events types include the following:

- *arrival* - Indicates tag is "new".
- *moved* - Indicates a tag has changed location.
- *departed* - Indicates a tag has "left".
- *returned* - Indicates a tag that was previously departed has shown up again.

NOTE: tag location changes generate moves if both sensors are in the same facility, otherwise they generate departure / arrival pairs.

4.6.4 Tag Processing Algorithm

The following pseudo code describes the logic flow that occurs for every tag read after tag location is recomputed based on the tag state. Event generation occurs for any step with event[EVENT_NAME] and tag state changes are indicated similarly by state[NEW_STATE].

```
UNKNOWN:
  is not POS ?
    event[ARRIVAL]
    state[PRESENT]

PRESENT:
  is POS ?
    is read time > inventory.POS.departed.threshold.millis ?
      event[DEPARTED]
      state[DEPARTED_POS]
    else
      is location changed ?
        is facility changed ?
          event[DEPARTED]
          event[ARRIVAL]
        else
          event[MOVED]
    else
      is EXIT ?
        is shedule ALL_ON or FROM_CONFIG ?
          state[EXITING]
      is location changed ?
        is facility changed ?
```

```

        event[DEPARTED]
        event[ARRIVAL]
    else
        event[MOVED]

EXITING:
    is POS ?
        is inventory.POS.departed.threshold.millis exceeded?
            event[DEPARTED]
            state[DEPARTED_POS]
        else is not EXIT and is location changed ?
            state[PRESENT]
            is facility changed ?
                event[DEPARTED]
                event[ARRIVAL]
            else
                event[MOVED]

DEPARTED_EXIT:
    is not POS ?
        is facility changed ?
            event[ARRIVAL]
        else
            event[RETURNED]
            state[PRESENT]
    is EXIT ?
        is shedule ALL_ON or FROM_CONFIG ?
            state[EXITING]

DEPARTED_POS:
    is not POS and is inventory.POS.return.threshold.millis exceeded ?
        is facility changed ?
            event[ARRIVAL]
        else
            event[RETURNED]
            state[PRESENT]
    is EXIT ?
        is shedule ALL_ON or FROM_CONFIG ?
            state[EXITING]

```


4.7 Upstream Events

The Gateway generates JSON RPC Notifications on the Upstream MQTT broker for inventory events and device alerts.

Table 3 RSP (RSP) Gateway Indications

Indication	Brief Description
inventory_event	Indicates that an inventory "event" has been detected. These indications are sent to the following MQTT topic < upstream mqtt topic prefix >/events
device_alert	Indicates an alert condition exists with either the Gateway or one of the connected Sensors. These indications are sent to the following MQTT topic < upstream mqtt topic prefix >/alerts

4.7.1 Inventory Event

Table 4 Inventory Event Notification Parameters

Parameter	Definition
sent_on	The millisecond timestamp of this indication.
data	An array of RFID tag records.

Each Data record contains the following fields.

facility_id	The ID assigned to the facility where the event occurred.
epc_code	The EPC of the tag associated with this record.
epc_encode_format	Reserved for future use, currently "tbd".
tid	The TID of the tag associated with this record.
event_type	Values are "arrival", "departed", "moved" and "returned".
timestamp	The millisecond timestamp of this event.
location	The associated sensor and antenna port for this tag.

```
{
  "jsonrpc": "2.0",
  "method": "inventory_event",
  "params": {
    "sent_on": 1424976117309,
    "data": [
      {
        "facility_id": "store-front",
        "epc_code": "100000000000000000002A1",
        "epc_encode_format": "tbd",
        "tid": "E28011606000020BCEC36DC1",
        "event_type": "arrival",
        "timestamp": 1424976117895,
        "location": "RSP-9b472d-1"
      }
    ]
  }
}
```

4.7.2 Gateway Device Alert

Table 5 Gateway Device Alert Parameters

Parameter	Definition
sent_on	The millisecond timestamp of this indication.
device_id	The ID assigned to the reporting RFID Sensor Platform.
alert_number	A unique number identifying the type of alert. The valid range of values is... 100 – RF_MODULE_ERR 101 – HIGH_AMBIENT_TEMP 102 – HIGH_CPU_TEMP 103 – HIGH_CPU_USAGE 104 – HIGH_MEMORY_USAGE 151 – DEVICE_MOVED 195 – RSP_CONNECTED 196 – RSP_SHUTTING_DOWN 197 – RSP_FACILITY_NOT_CONFIGURED 198 – RSP_LOST_HEARTBEAT 199 – RSP_LAST_WILL_AND_TESTAMENT 240 – GATEWAY_STARTED 241 – GATEWAY_SHUTTING_DOWN
alert_description	A corresponding human readable text description.
severity	A prioritized severity level of the alert. The valid range of values is... <i>"info", "warning", "urgent", and "critical".</i>
optional	A series of optional number or string parameters providing further information about the alert.
facilities	A list of facility_id's that may be affected by this alert.

```
{
  "jsonrpc": "2.0",
  "method": "device_alert",
  "params": {
    "sent_on": 1424976117309,
    "device_id": "RSP-abcdef",
    "alert_number": 101,
    "alert_description": "HighAmbientTemp",
    "severity": "warning",
    "optional": {
      "string": "degrees_c",
      "number": 51
    }
  },
  "facilities": [
    "store-front"
  ]
}
```

5 Build and Deploy

The source code and more information regarding the Gateway can be downloaded from the Intel® Open Source Portal <https://01.org/rsp-sw-toolkit>. The project sources are located on Github at <https://github.com/intel/rsp-sw-toolkit-gw>.

The Linux instructions can also be found in the Github README.md file. They assume an Ubuntu 18.04 LTS installation. This section will cover both Linux and Windows® 10 installation procedures.

5.1 Linux

5.1.1 Install Dependencies

```
# development dependencies
sudo apt-get install default-jdk git gradle

# runtime dependencies
sudo apt-get install mosquitto avahi-daemon ntp ssh
```

5.1.2 Clone the Project

```
# the cloned repo can be anywhere but as a suggestion,
# create a projects directory
mkdir -p ~/projects
cd ~/projects
git clone https://github.com/intel/rsp-sw-toolkit-gw.git
```

5.1.3 Build and Deploy

```
# build an archive suitable for deployment
cd ~/projects/rsp-sw-toolkit-gw/
gradle deploy
```

5.2 Windows® 10

5.2.1 Install Dependencies

Download and install Open JDK for Windows®. <https://openjdk.java.net/>

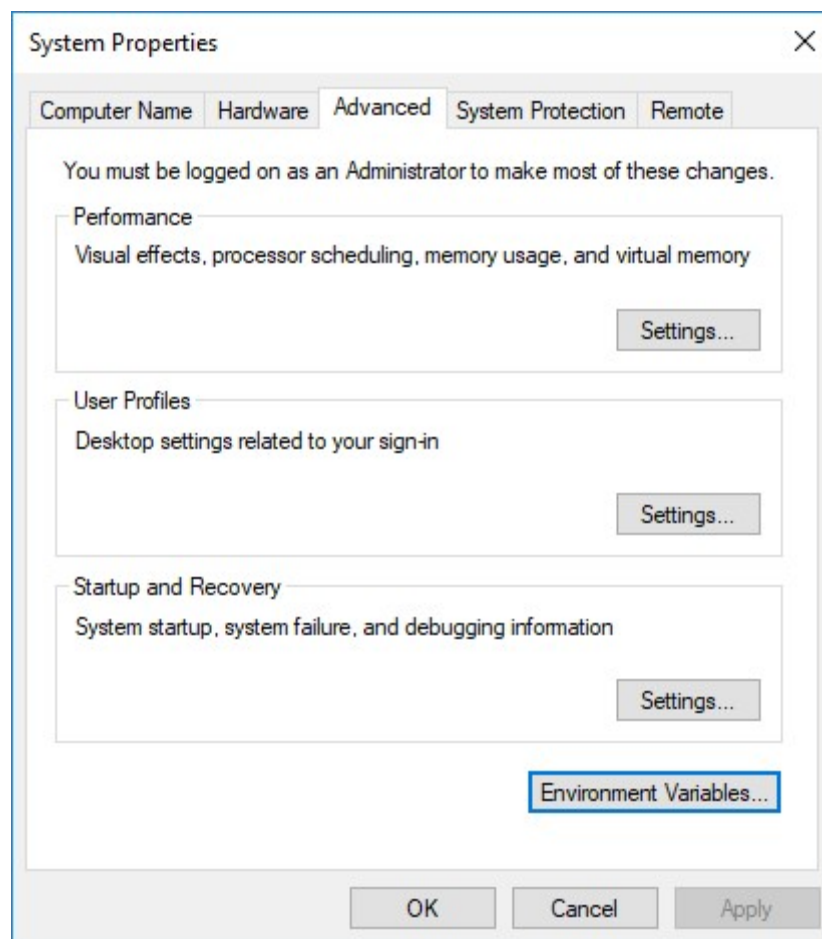
Download and install OpenSSL for Windows®. An easy way of obtaining OpenSSL without running into a risk of installing unknown software from 3rd party websites, is by using the openssl.exe that comes with the “Git for Windows®” installation. It can be found here.

C:\Program Files\Git\usr\bin\openssl.exe

5.2.2 Edit the Path Environment Variable

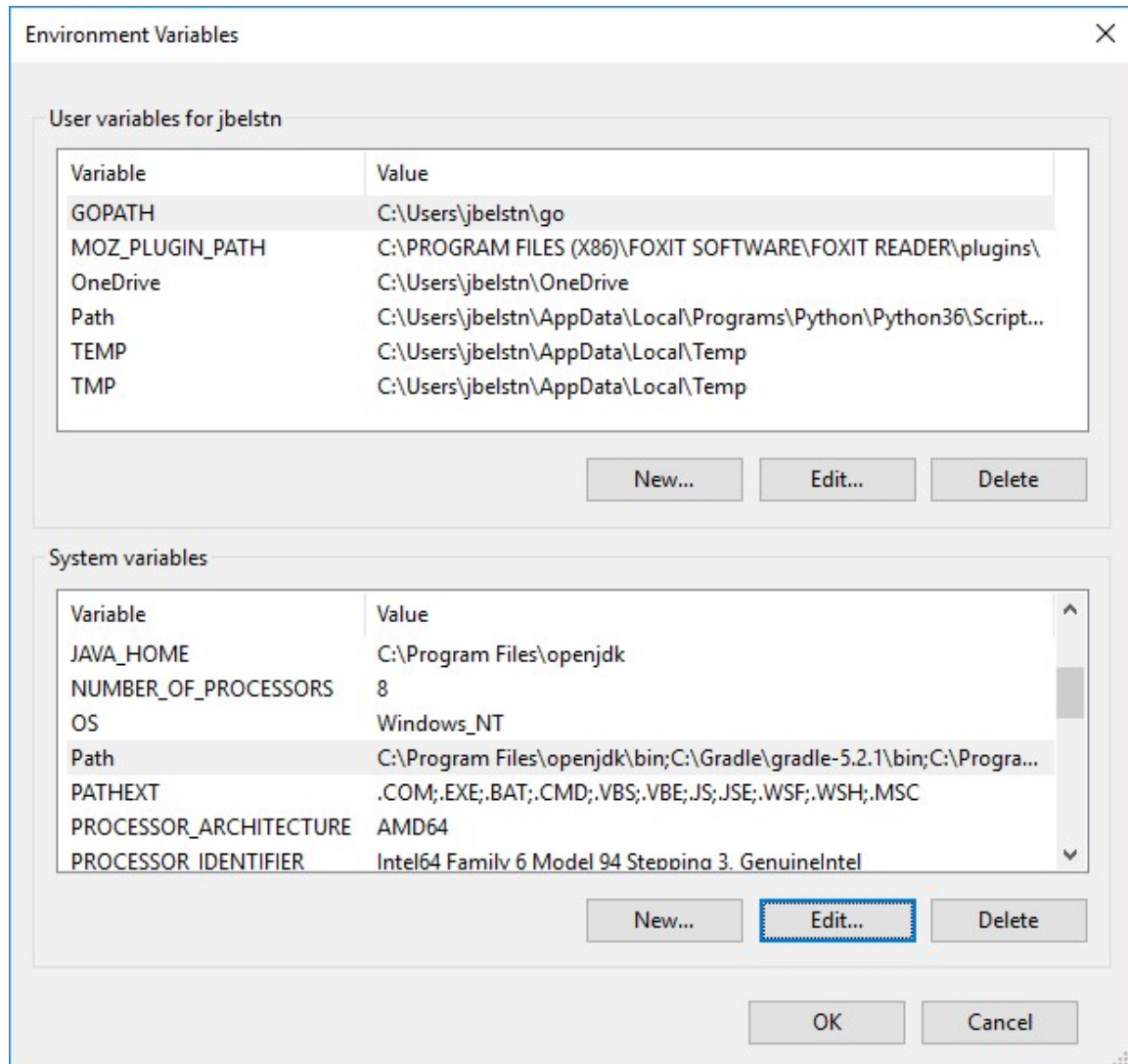
Add the paths to both the Open JDK bin directory and to the OpenSSL bin directory to your “Path” environment variable as shown below.

Open the System Properties window and select “Environment Variables...”

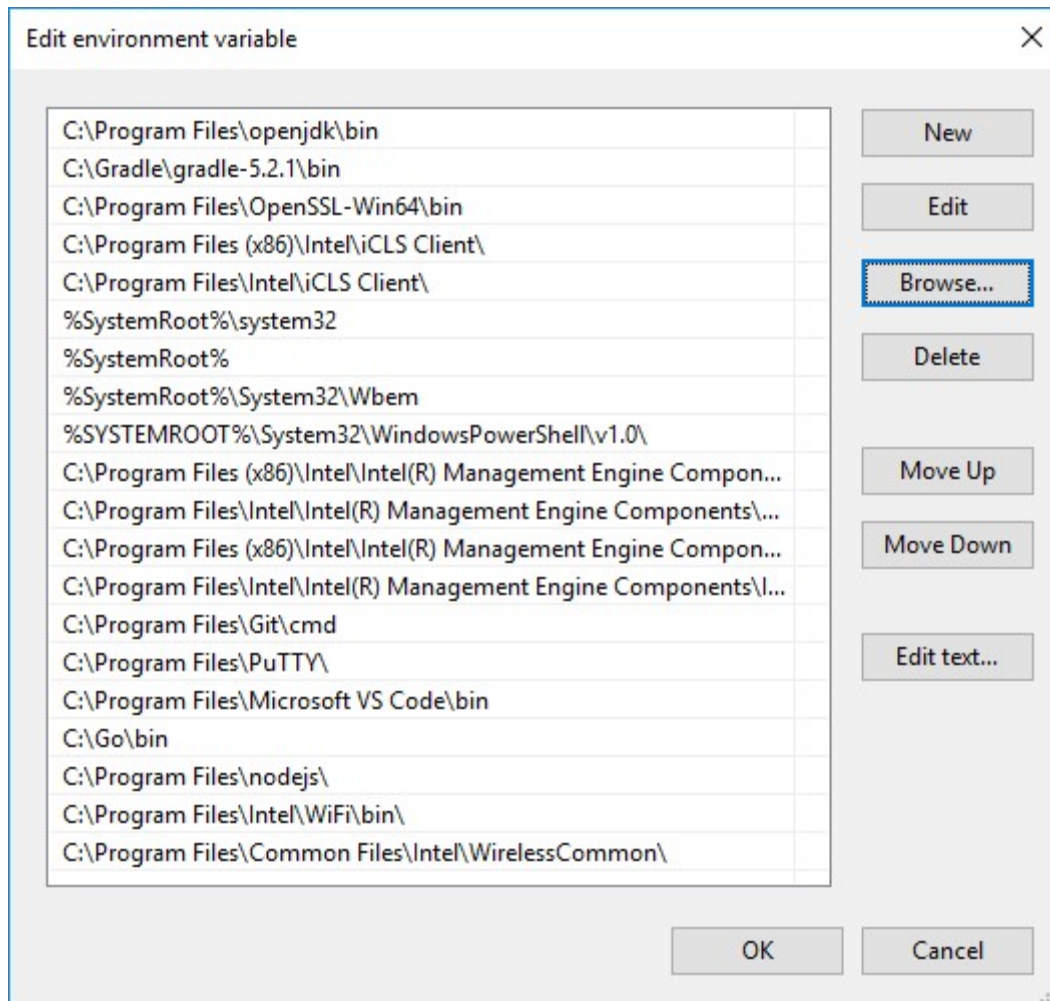


Build and Deploy

Highlight “Path” under System variables and press “Edit...”

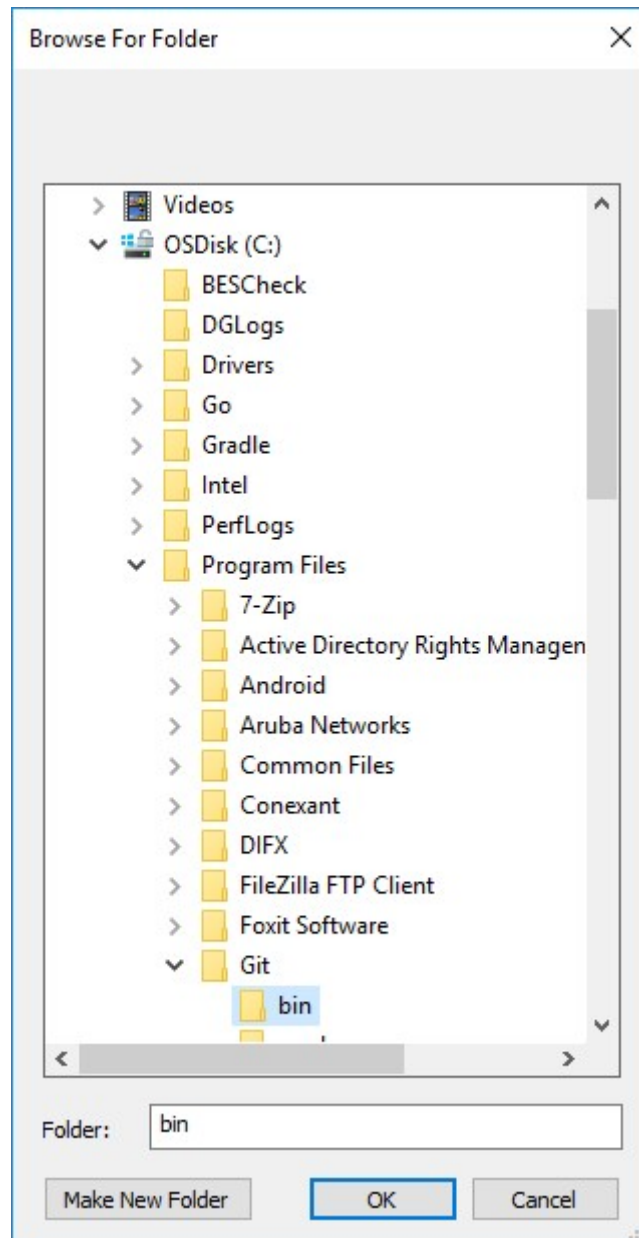


Select "Browse..."



Build and Deploy

Locate the desired directory and press "OK"



5.2.3 Download the Project

Download the zip file from the Github repository as shown below.

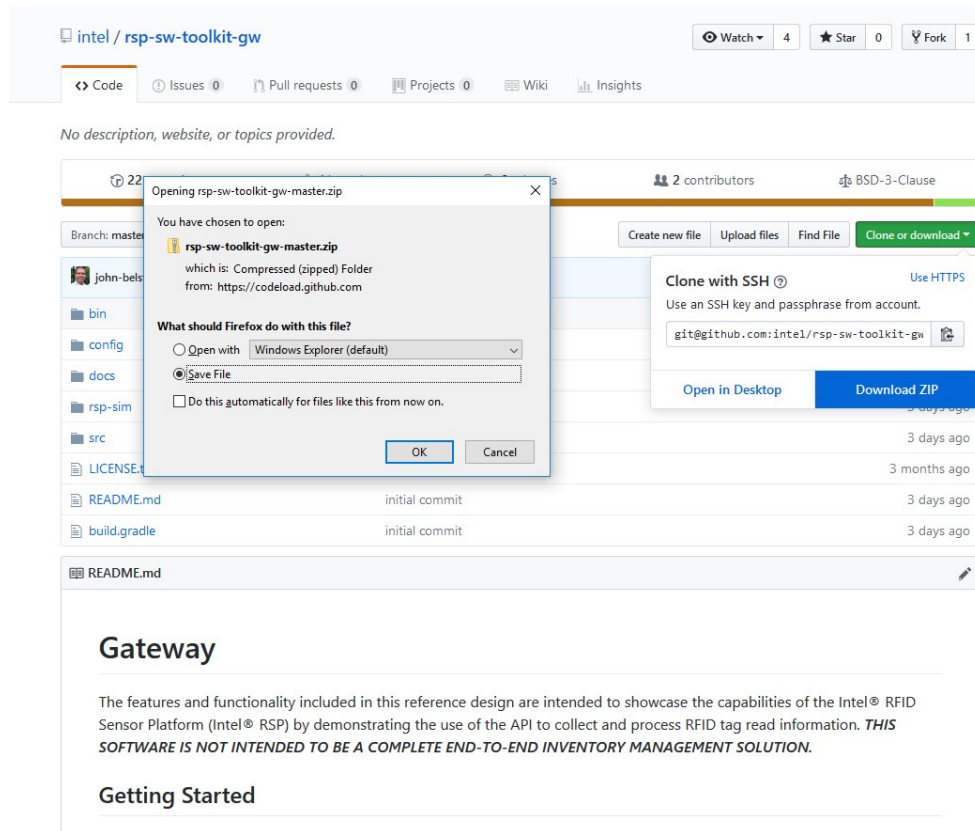
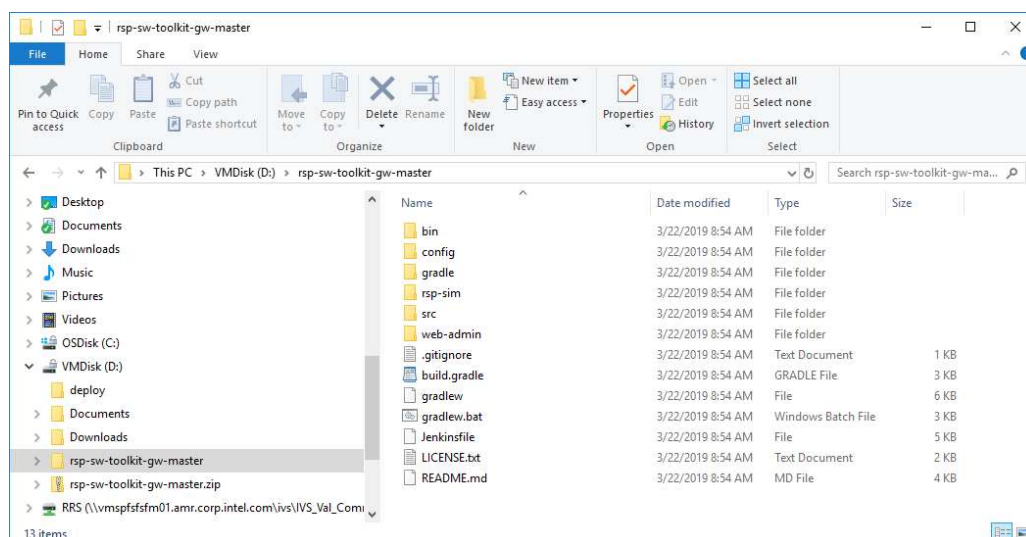


Figure 5 Downloading a Zip Version of the Project

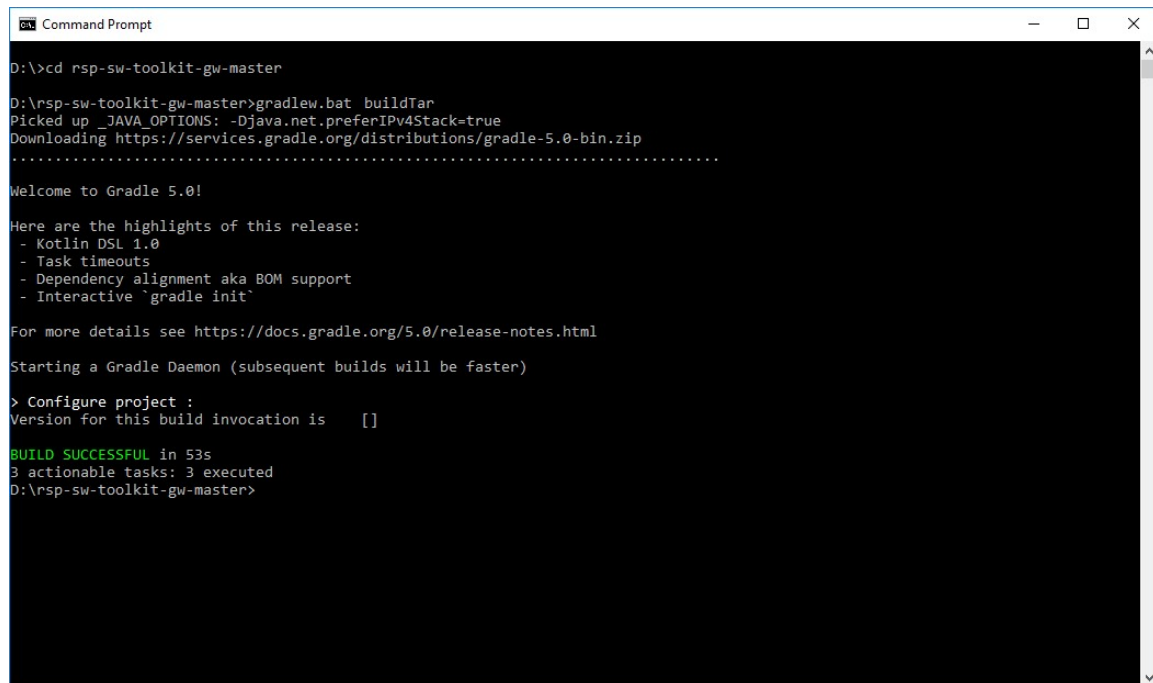
Extract the zip file into a working directory.



Build and Deploy

5.2.4 Build and Deploy

From the top of the Project directory, execute “gradlew.bat buildTar”. Then extract the .tar file in the “deploy” directory.



```
D:\>cd rsp-sw-toolkit-gw-master

D:\rsp-sw-toolkit-gw-master>gradlew.bat buildTar
Picked up _JAVA_OPTIONS: -Djava.net.preferIPv4Stack=true
Downloading https://services.gradle.org/distributions/gradle-5.0-bin.zip
.....

Welcome to Gradle 5.0!

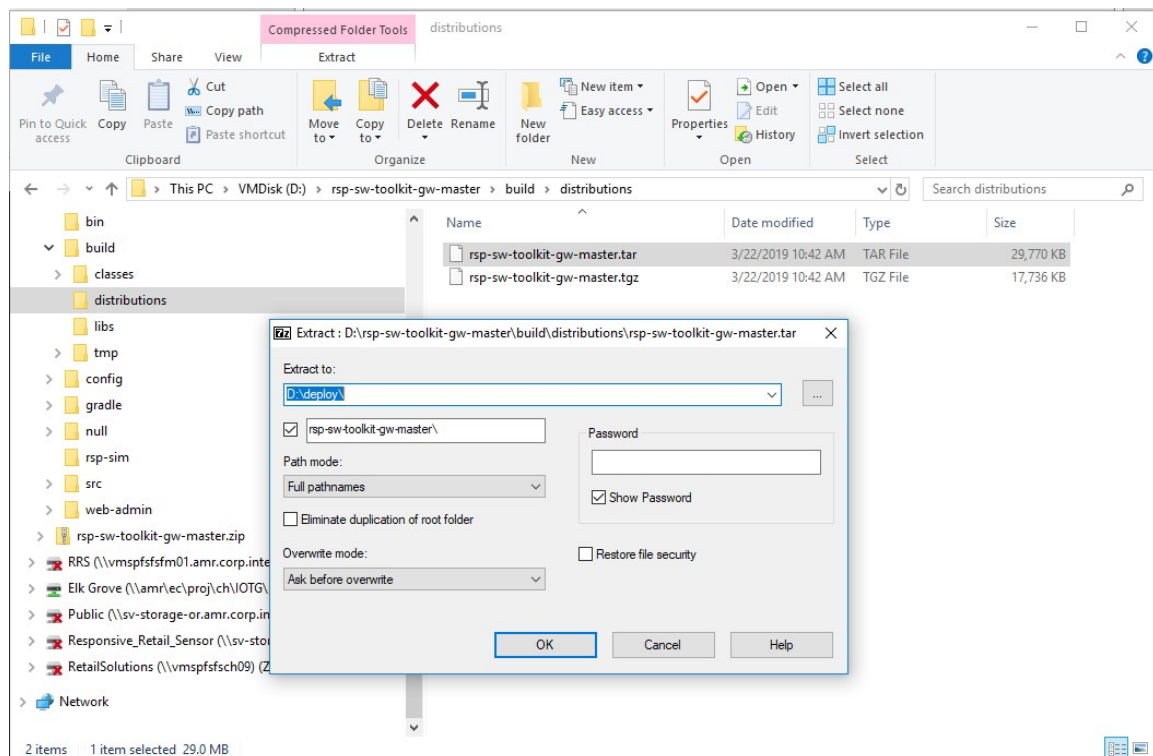
Here are the highlights of this release:
- Kotlin DSL 1.0
- Task timeouts
- Dependency alignment aka BOM support
- Interactive `gradle init`

For more details see https://docs.gradle.org/5.0/release-notes.html

Starting a Gradle Daemon (subsequent builds will be faster)

> Configure project :
Version for this build invocation is []

BUILD SUCCESSFUL in 53s
3 actionable tasks: 3 executed
D:\rsp-sw-toolkit-gw-master>
```



5.3 Project Runtime Folders

The following is a list of the top level folders of a deployment.

path in this example: ~/deploy/gateway/

- cache - the contents of this folder are intended to support maintaining the runtime state of the application. For example, the inventory manager will write a file to preserve the state of the inventory across restarts of the gateway. The same goes for schedule manager and other app components.
- config
 - behaviors - Behavior configuration files in this directory are available for use in the application. NOTE: the behavior id should match the filename.
 - mobility - The location for mobility profile configurations.
- exported-tokens - Location for the gateway to write out tokens used in provisioning Sensors with mutual authentication
- lib - Location of java libraries. All .jar files in this directory are available in the runtime classpath.
- log - Location for gateway logs
- sensor-sw-repo - Location of the sensor's software package repo files
- snapshot - Location for inventory snapshots.
- stats - Location for tag and Sensor statistics
- tagread - Location for raw tag read files (when enabled)
- web-admin - Location for web-admin resources

5.4 Configure

The configuration file located at “~/deploy/rsp-sw-toolkit-gw/config/gateway.cfg”. For initial testing, it is generally not necessary to modify this file from its defaults.

```
#-----
#-- GATEWAY configuration
#--
#-- Default values used by the application are included
#-- here but commented out as reference (except for ntp)
#--
#-- Configuring Hosts:
#-- For configuration items that refer to a host,
#-- leaving that entry blank will have the effect
#-- of using the server hostname and the expectation
#-- that the corresponding service is available on
#-- the same host as the gateway
#-----

#-- GATEWAY
#--
# gateway.device_id =
#-----

#-- NTP
#--
ntp.server.host = pool.ntp.org
#-----

#-- MQTT
#--
#-- NOTE: that supported protocols are either 'tcp' or 'ssl'
#--
#-- NOTE: if used, the 'downstream' password is also sent to
#-- the sensors and they will attempt to connect to the
#-- broker using their device id and this password.
#--
# mqtt.downstream.protocol = tcp
# mqtt.downstream.host =
# mqtt.downstream.port = 1883
# mqtt.downstream.username =
# mqtt.downstream.password =
#--
# mqtt.upstream.protocol = tcp
# mqtt.upstream.host =
# mqtt.upstream.port = 1883
# mqtt.upstream.username =
# mqtt.upstream.password =
#-----

#-- RSP SOFTWARE PACKAGE REPO
#--
#-- The RSP is sent this information and will attempt to update itself
#-- using the packages in this repo.
#-- NOTE: the packages must be signed by Intel to be installed.
#--
# repo.rsp.protocol = http
# repo.rsp.host =
# repo.rsp.port = 8080
# repo.rsp.archs = all,armv7at2hf-neon,armv7at2hf-neon-mx6qdl,hx000
#-----

#-- CONSOLE
#--
```

```

#-- The ssh connection parameters used to log in to the gateway CLI
#--
# console.port = 5222
# console.userid = gwconsole
# console.password = gwconsole
#-----

#-----
#-- INVENTORY
#--
#-- Ageout: Tags that have not been read in this amount of time
#-- will be removed from the inventory database.
#--
# inventory.ageout.hours = 336
#--
#-- Departed: Tags that have been read by an EXIT sensor will
#-- generate a departure if they have not been read by any sensor
#-- for this amount of time
#--
# inventory.aggregate.departed.threshold.millis = 30000
#-----

#-----
#-- POINT OF SALE
#--
#-- Departed: the amount of time that a tag must be in the inventory before
#-- it can depart from a POS sensor
#-- default is 1 hour (1 * 60 * 60 * 1000)
#--
# inventory.POS.departed.threshold.millis = 3600000
#--
#-- Returned: for tags that have departed by POS,
#-- this time threshold must be exceeded before the tag will be
#-- 'returned' to inventory
#-- default is 1 day (24 * 60 * 60 * 1000)
#--
# inventory.POS.return.threshold.millis = 86400000
#-----

#-----
#-- PROVISIONING
#--
#-- Root CA Cert: if configured as https, the sensor uses Debian
#-- ca cert store for authentication
#--
# provision.ca.cert.protocol = http
#--
#-- Sensor Credentials: if configured as https, the sensor authenticates
#-- this connection using the downloaded root CA
#--
# provision.sensor.credentials.protocol = https
#--
#-- Ports used by the local REST server for the certificate and credentials
#--
# provision.http.port = 8080
# provision.tls.port = 8443
#--
#-- Sensor Token: Indicates whether mutual authentication is required
#-- between the sensor and gateway upon connection.
#-- For more details, please consult the user guide.
#--
# provision.sensor.token.required = false
#
#-- password for the pcs12 java keystore
provision.keystore.password = server123store456
#-----

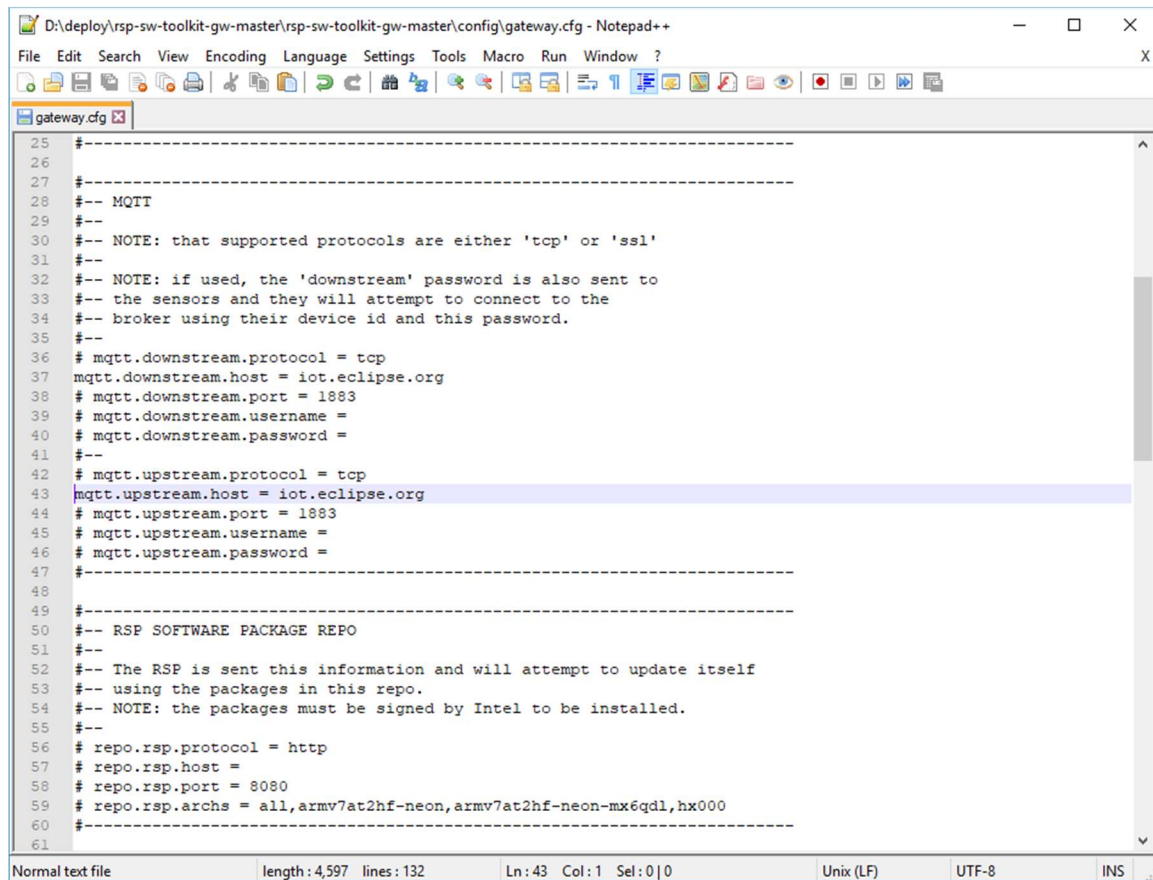
```

Build and Deploy

5.4.1 Windows® 10

Rather than downloading and installing a separate Windows MQTT Server, change the “gateway.cfg” file to use one of the public servers. NOTE: Use for Testing Purposes Only!

The example below shows changing both the downstream and upstream MQTT host to use the “iot.eclipse.org” public server.



```
D:\deploy\rsp-sw-toolkit-gw-master\rsp-sw-toolkit-gw-master\config\gateway.cfg - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Window ?
gateway.cfg
25 #-----
26
27 #-----
28 #-- MQTT
29 #--
30 #-- NOTE: that supported protocols are either 'tcp' or 'ssl'
31 #--
32 #-- NOTE: if used, the 'downstream' password is also sent to
33 #-- the sensors and they will attempt to connect to the
34 #-- broker using their device id and this password.
35 #--
36 # mqtt.downstream.protocol = tcp
37 mqtt.downstream.host = iot.eclipse.org
38 # mqtt.downstream.port = 1883
39 # mqtt.downstream.username =
40 # mqtt.downstream.password =
41 #--
42 # mqtt.upstream.protocol = tcp
43 mqtt.upstream.host = iot.eclipse.org
44 # mqtt.upstream.port = 1883
45 # mqtt.upstream.username =
46 # mqtt.upstream.password =
47 #-----
48
49 #-----
50 #-- RSP SOFTWARE PACKAGE REPO
51 #--
52 #-- The RSP is sent this information and will attempt to update itself
53 #-- using the packages in this repo.
54 #-- NOTE: the packages must be signed by Intel to be installed.
55 #--
56 # repo.rsp.protocol = http
57 # repo.rsp.host =
58 # repo.rsp.port = 8080
59 # repo.rsp.archs = all,armv7at2hf-neon,armv7at2hf-neon-mx6qdl,hx000
60 #-----
61
Normal text file      length: 4,597  lines: 132      Ln: 43  Col: 1  Sel: 0 | 0      Unix (LF)      UTF-8      INS
```

5.5 Certificate Generation

The Gateway supports encrypted communication protocols for both Sensor provisioning via REST endpoints and Sensor data communications via MQTT. In order to support this, a root certificate and a properly signed server certificate are required. The shell script `gen_keys.sh` can be used to generate the keys and certificates required. The script uses various `openssl` and `keytool` operations in order to generate a set of certificates and a keystore for use by the Gateway and the Sensor.

There are several files created by the `gen_keys.sh` script but only the following are used by the Gateway.

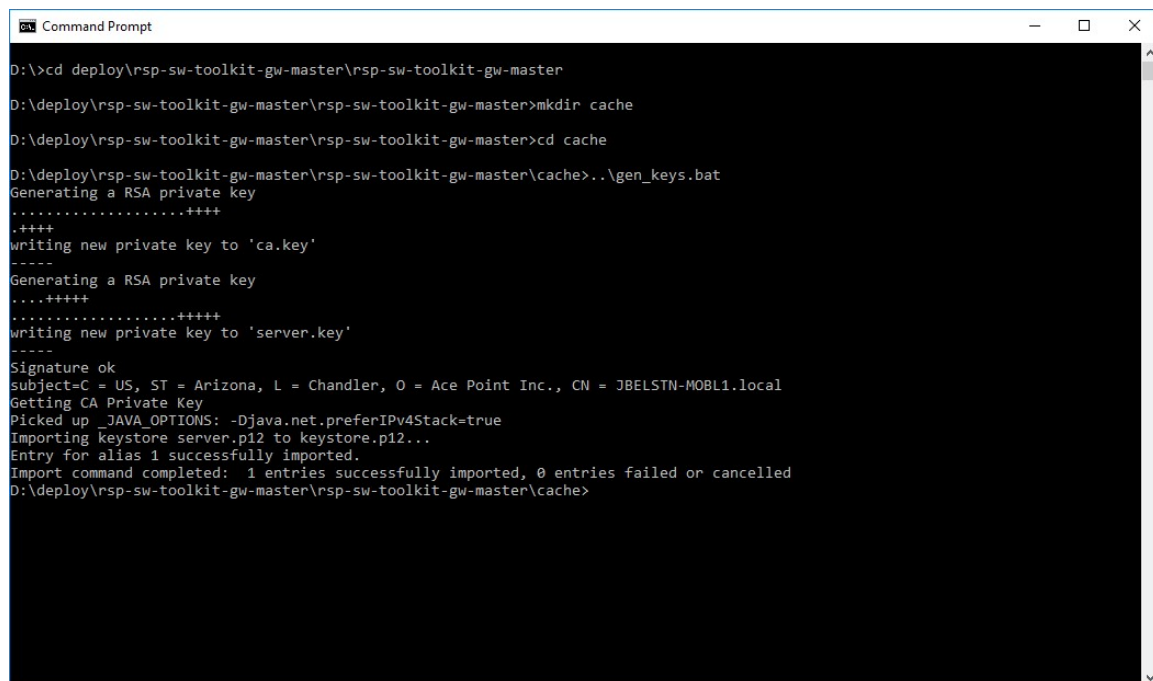
- *ca.crt* - the root CA public certificate
- *server.crt* - a certificate signed by the root CA
- *keystore.p12* - a pkcs12 java keystore for the server certificate, used by the Jetty REST endpoints of the Gateway.

Execute the script to create these files.

5.5.1 Linux

```
mkdir -p ~/deploy/rsp-sw-toolkit-gw/cache
cd ~/deploy/rsp-sw-toolkit-gw/cache
~/deploy/rsp-sw-toolkit-gw/gen_keys.sh
```

5.5.2 Windows® 10



```

D:\>cd deploy\rsp-sw-toolkit-gw-master\rsp-sw-toolkit-gw-master
D:\deploy\rsp-sw-toolkit-gw-master\rsp-sw-toolkit-gw-master>mkdir cache
D:\deploy\rsp-sw-toolkit-gw-master\rsp-sw-toolkit-gw-master>cd cache
D:\deploy\rsp-sw-toolkit-gw-master\rsp-sw-toolkit-gw-master\cache>..\gen_keys.bat
Generating a RSA private key
.....+++++
.++++
writing new private key to 'ca.key'
-----
Generating a RSA private key
.....+++++
.....+++++
writing new private key to 'server.key'
-----
Signature ok
subject=C = US, ST = Arizona, L = Chandler, O = Ace Point Inc., CN = JBELSTN-MOBL1.local
Getting CA Private Key
Picked up _JAVA_OPTIONS: -Djava.net.preferIPv4Stack=true
Importing keystore server.p12 to keystore.p12...
Entry for alias 1 successfully imported.
Import command completed: 1 entries successfully imported, 0 entries failed or cancelled
D:\deploy\rsp-sw-toolkit-gw-master\rsp-sw-toolkit-gw-master\cache>

```

5.6 Run

Start the Gateway application as shown below.

5.6.1 Linux

```
~/deploy/rsp-sw-toolkit-gw/run.sh
```

5.6.2 Windows® 10

```
c:  
cd ~\deploy\rsp-sw-toolkit-gw  
~\deploy\rsp-sw-toolkit-gw\run.bat
```

5.7 Sensor Connection Sequence

Once power-on and boot is complete, the Sensor uses the information contained in the gateway mDNS announcement to synchronize time via NTP and then download the root certificate. If the gateway is configured to require the use of provisioning tokens, the sensor must have been provisioned with an NFC provisioning tag that contains a matching sha256 hash of the root certificate. Then the sensor sends a post to the gateway's sensor credentials endpoint to retrieve the MQTT connection credentials. If provisioning token is required, the token value is included in the post request and the gateway refuses to provide credentials if the token is invalid. The sensor connects to the downstream MQTT broker instance and sends a connect request to the gateway. The gateway responds with a connect response and the sensor signals that it is "ready". At this point, the sensor is connected.

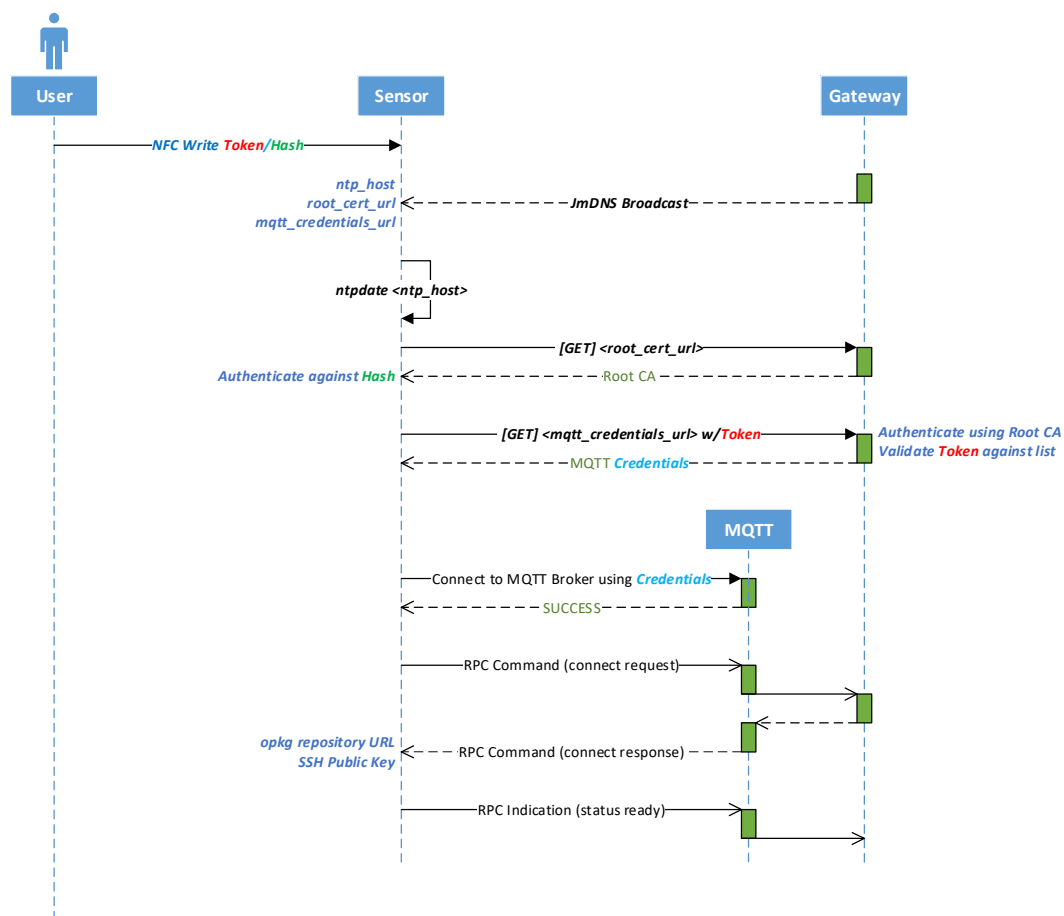


Figure 6 Discovery and Mutual Authentication Process

6 System Provisioning Cluster Tokens

As described in the previous section, the gateway optionally supports a mutual authentication scheme with Sensors by requiring provisioning tokens. To enable mutual authentication, set the following configuration variable in the file `gateway.cfg` to "true".

```
provision.sensor.token.required = true
```

When this feature is enabled, the "Sensor NFC App" must be used to program provisioning credentials into a Sensor before the Sensor connects to the Gateway. The credentials are programmed into the Sensor via an NFC capable device. See the "Intel® RSP SW Toolkit – Sensor NFC App" documentation for detailed instructions on programming the Sensor.

Leveraging the tokens used for authentication, clusters can have their sensor groups replaced by a single token. In this case, after the gateway validates the token, it maps that sensor to the corresponding cluster configuration via the token reference. Adopting this practice supports enterprise scalability as a single cluster configuration can be used across gateway installations. When the sensors are installed, the NFC App is used to essentially "program" the sensor with the facility, personality, and scheduling parameters. A unique token is required for each sensor group.

The previous cluster configuration is shown here using tokens instead of sensor groups.

git repo: `src/test/resources/clusters/sample_provisioning_cluster_config.json`

```
{
  "id": "SampleProvisioningClusterConfig",
  "clusters": [
    {
      "id": "BackStockDefault",
      "facility_id": "BackStock",
      "behavior_id": "DefaultDeepScan",
      "tokens": [
        {
          "username": "BS_DEFAULT_001",
          "token": "BackStockDeepScan001UUWWYYBBDDFFE3DA5098692CE27945AA367189B52C58",
          "generatedTimestamp": 0,
          "expirationTimestamp": -1
        },
        {
          "username": "BS_DEFAULT_002",
          "token": "BackStockDeepScan002UUWWYYBBDDFFE3DA5098692CE27945AA367189B52C58",
          "generatedTimestamp": 0,
          "expirationTimestamp": -1
        }
      ]
    }, {
      "id": "SalesFloorDefault",
      "facility_id": "SalesFloor",
      "behavior_id": "DefaultMobility",
      "tokens": [
        {
          "username": "SF_DEFAULT_001",
          "token": "SalesFloor01Default001WWYYBBDDFFE3DA5098692CE27945AA367189B52C58",
          "generatedTimestamp": 0,
          "expirationTimestamp": -1
        }
      ]
    }
  ]
}
```

```

        "username": "SF_DEFAULT_002",
        "token": "SalesFloor01Default002WWYYBBDDFFE3DA5098692CE27945AA367189B52C58",
        "generatedTimestamp": 0,
        "expirationTimestamp": -1
    }
]
}, {
    "id": "SalesFloorExit",
    "facility_id": "SalesFloor",
    "behavior_id": "DefaultExit",
    "personality": "EXIT",
    "tokens": [
        {
            "username": "SF_EXIT_001",
            "token": "SalesFloor01Exit001SUUWWYYBBDDFFE3DA5098692CE27945AA367189B52C58",
            "generatedTimestamp": 0,
            "expirationTimestamp": -1
        }
    ]
}, {
    "id": "SalesFloorPOS",
    "facility_id": "SalesFloor",
    "behavior_id": "DefaultPOS",
    "personality": "POS",
    "tokens": [
        {
            "username": "SF_POS_001",
            "token": "SalesFloor01POS001TSUUWWYYBBDDFFE3DA5098692CE27945AA367189B52C58",
            "generatedTimestamp": 0,
            "expirationTimestamp": -1
        }
    ]
}, {
    "id": "SalesFloorFittingRoom",
    "facility_id": "SalesFloor",
    "behavior_id": "DefaultFittingRoom",
    "personality": "FITTING_ROOM",
    "tokens": [
        {
            "username": "SF_FITTING_ROOM_001",
            "token": "SalesFloor01FittingRoom001BBDDFFE3DA5098692CE27945AA367189B52C58",
            "generatedTimestamp": 0,
            "expirationTimestamp": -1
        }
    ]
}
]
}
}

```

The credentials include a sha256 hash of the root certificate and the token value of a provisioning token.

The certificate hash is used by the Sensor to verify the root CA certificate that the Sensor downloads as part of the connection sequence. If using the `gen_keys` script provided

System Provisioning Cluster Tokens

with the toolkit, the root CA certificate will be located in the deployment directory under the cache folder `~/deploy/gateway/cache/ca.crt`.

The parameters of a provisioning token are

- **username:** string used only as reference for the token.
- **token:** 64 character ASCII utf-8 character string. stored on sensor, transmitted to gateway in the connect request if tokens are required by the gateway.
- **generatedTimestamp:** seconds since the epoch.
- **expirationTimestamp:** seconds since the epoch, gateway will reject a connection attempt from sensors if this timestamp is invalid.

After a cluster configuration is loaded into the gateway, the tokens needed for the NFC Sensor App can be extracted to the exported tokens folder.

```
rfid-gw> clusters load.file /home/userid/projects/rsp-sw-  
toolkit/gateway/src/test/resources/clusters/sample_provisioning_cluster_config.json
```

```
-----  
completed  
-----
```

```
rfid-gw> clusters export.tokens
```

```
-----  
exported: /home/userid/deploy/gateway/exported-tokens/token_SF_DEFAULT_002.json  
exported: /home/userid/deploy/gateway/exported-tokens/token_SF_POS_001.json  
exported: /home/userid/deploy/gateway/exported-tokens/token_BS_DEFAULT_002.json  
exported: /home/userid/deploy/gateway/exported-tokens/token_BS_DEFAULT_001.json  
exported: /home/userid/deploy/gateway/exported-tokens/token_SF_FITTING_ROOM_001.json  
exported: /home/userid/deploy/gateway/exported-tokens/token_SF_DEFAULT_001.json  
exported: /home/userid/deploy/gateway/exported-tokens/token_SF_EXIT_001.json  
-----
```

7 Web Administration

A web based administration interface is enabled on the gateway. If running the gateway locally, the following URL can be used. The port is determined by the provision ports in the gateway configuration file.

<http://127.0.0.1:8080/web-admin/>

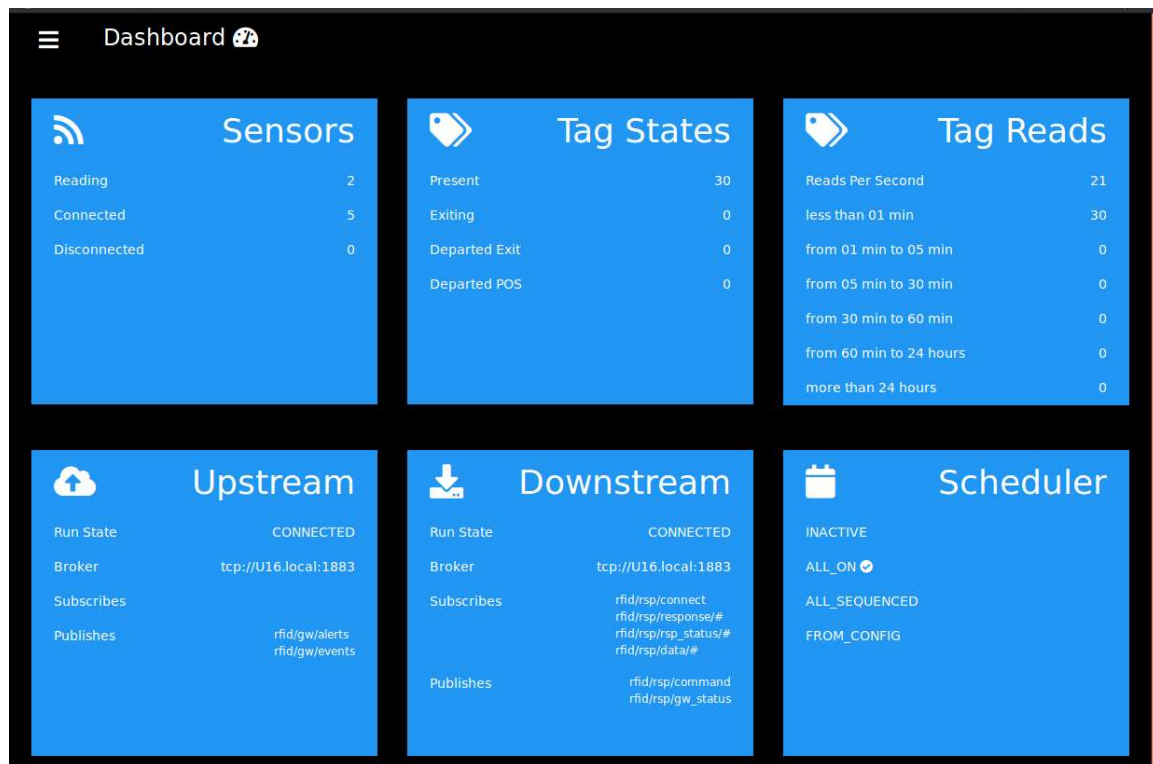


Figure 7 Web Admin Dashboard

The Web Admin Dashboard and associated pages provide a friendly way to get sensor status and tag information from the Gateway as well as control the RFID scheduling. The goal is to expose nearly all of the CLI functionality from the Web Admin.

8 Command Line Interface (CLI)

The Command Line Interface (CLI) provides a means of interacting with the Gateway to configure and manage the Sensors. The CLI is accessed using a secure shell login from either a Linux terminal or Windows® PuTTY session. Check the gateway configuration file for the username, password, and port number.

The CLI supports auto command completion at any point by pressing the <tab> key to complete the command or to view a list of available commands or options. Pressing <enter> prior to the completion of a command will display command usage and parameter definitions. Generous use of the <tab> and <enter> keys is the best way to learn the available options of the CLI.

```
$ ssh -p5222 gwconsole@localhost
Password authentication *****
```

RFID Gateway console session

<tab> to view available commands
'clear' to clear the screen/console
'quit' to end

rfid-gw>

exit	quit	scheduler	system	upstream	downstream	log	sensor
inventory	version	clusters					

8.1 exit - quit

Causes the CLI console session to terminate. Does not affect the running Gateway application.

8.2 scheduler

USAGE

```
> scheduler info <topic>
    Displays info about the topic

> scheduler show
    Displays info about current state of the schedule manager

> scheduler activate.all.on
    Transitions to all sensors reading tags at the same time

> scheduler activate.all.sequenced
    Transitions to each sensor reading tags one at a time

> scheduler activate.from.config
    Transitions to running per the existing cluster configuration

> scheduler deactivate
    Deactivates any scheduling activities and causes the sensors to stop reading
```

8.3 system

----- USAGE

```
> system info
    Displays processor and memory information
```

8.4 upstream

----- USAGE

```
> upstream show
    Show the status and topics of the Upstream MQTT
```

8.5 downstream

----- USAGE

```
> downstream show
    Show the status and topics of the Downstream MQTT
```

8.6 log

----- USAGE

```
> log show
    Shows the current log level settings for all active loggers

> log set <log id> <level>
    Sets the level of the logger associated with the log_id
```

8.7 sensor

NOTE: the scheduler DOES NOT account for individually starting or stopping sensor reading from the CLI. Recommend setting scheduler INACTIVE.

----- USAGE

```
> sensor show [ regex ]
    Displays a list of all known sensors
    Optional regex will match against the sensor id
    A sensor is known to the gateway by the following:
        Listed in facility group configuration
        Listed in personality group configuration
        Listed in schedule group configuration
    Actively connects to the gateway regardless of facility, personality, or schedule
inclusion

> sensor start.reading <device_id>...
```

Command Line Interface (CLI)

```
Start reading using the currently configured behavior(s)

> sensor stop.reading <device_id>...
    Stop reading

> sensor reset <device_id>...
    Command device to perform a reset

> sensor reboot <device_id>...
    Command device to perform a reboot

> sensor shutdown <device_id>...
    Command device to perform an OS shutdown
    CAUTION: Do this ONLY prior to physically removing the device!

> sensor remove <device_id>...
    Removes device from ALL configurations (facility, personality, schedule)
    The device must be in the DISCONNECTED state

> sensor get_bist_results <device_id>...
    Retrieve Built-In-Self-Test results

> sensor get.last.comms <device_id>...
    Display last heard time

> sensor get_state <device_id>...
    Retrieve device capabilities

> sensor get_sw_version <device_id>...
    Retrieve device software version

> sensor set.alert.threshold <type> <severity> <device_id>...
    Change a Device Alert Threshold

> sensor set.behavior <behavior> <device_id>...
    Apply a predefined RFID behavior

> sensor set.facility <facility_id> <device_id>...
    Assign the Facility ID

> sensor set.personality <personality> <device_id>...
    Add a personality

> sensor clear.personality <device_id>...
    Clears a personality

> sensor set.led <led_state> <device_id>...
    Apply a visual indicator behavior

> sensor set.motion send.events <true|false>capture.images <true|false> <device_id>...
    Enable or disable the sending of Motion Events
    Enable or disable the capture of .jpg images

> sensor ack.alert <alert_type> <true|false> <device_id>...
    Acknowledge a Device Alert

> sensor mute.alert <alert_type> <true|false> <device_id>...
    Disable a particular Device Alert

> sensor tokens <device_id>...
    Show the token associated with the device
-----
```

8.8 inventory

----- USAGE

```

> inventory summary
    Shows the tags in summarized groupings

> inventory update.with </full/path/to/tag-scan-file.csv>
    Updates the current inventory with data from the specified csv file

> inventory replace.with </full/path/to/tag-scan-file.csv>
    Replaces the current inventory with data from the specified csv file

> inventory detail [ regex ]
    Shows details of each tag in the database
    Optional regex will match against the EPC

> inventory exiting
    Shows the tags in the exiting table, potential departed tags

> inventory snapshot
    Writes a snapshot of the current inventory to file

> inventory unload
    !! WARNING !! WARNING !! WARNING !! WARNING !! WARNING !! WARNING !!
    Clears all tag reads from the gateway including the cached file. No recovery available
    This will cause new arrival events to be generated for all tags

> inventory stats show [ regex ]
    Shows read statistics of each tag in the database
    Optional regex will match against the EPC

> inventory stats snapshot [ regex ]
    Writes a snapshot of tag read statistics to file
    Optional regex will match against the EPC

> inventory stats set.regex
    Sets a regular expression used for filtering stats during recording

> inventory stats check.regex
    Displays the stats the current regex will return and also shows the regex pattern

> inventory stats start.recording
    Starts taking stats sanpshots at a regular interval (5 seconds)

> inventory stats stop.recording

> inventory mobility.profile show
    Shows the mobility profile settings used in the moved event algorithm

> inventory mobility.profile set
    Sets the mobility profile used in the moved event algorithm

> inventory waypoints show
    Shows the tags waypoint history

> inventory waypoints snapshot
    Writes a snapshot of tag waypoints to file
    -----

```


8.9 version

USAGE

```
> version info
    Displays the software version information
```

8.10 clusters

USAGE

```
> clusters show
    Displays info about current state of the schedule manager

> clusters load.file
    Load a cluster configuration from the specified JSON file path

> clusters show.tokens
    Displays the tokens included in the configuration

> clusters export.tokens
    Exports the tokens included in the configuration as separate json files
```

9 Sensor Software Update

Sensors utilize 'opkg' software package management to keep their software updated with an external package repository and the gateway can host this repository. The only thing needed is to install the repository files into the appropriate directory.

```

$ tar -xf hx000-rrs-repo-19.1.2.7-5-gce5ed4f.tgz
$ mv ./hx000-rrs-repo-19.1.2.7-5-gce5ed4f/* ~/deploy/rsp-sw-toolkit-gw/sensor-sw-repo/
$ cd ~/deploy/rsp-sw-toolkit-gw/sensor-sw-repo
$ ls -la
total 60
drwxr-xr-x  5 user user 4096 Mar 21 16:31 .
drwxr-xr-x 14 user user 4096 Mar 21 15:21 ..
drwxr-xr-x  2 user user 4096 Mar 13 11:20 all
drwxr-xr-x  2 user user 24576 Mar 13 11:20 armv7at2hf-neon
drwxr-xr-x  2 user user 20480 Mar 13 11:23 hx000
-rw-r--r--  1 user user  0 Mar 13 11:14 Packages
-rw-r--r--  1 user user 3134 Mar 13 11:14 Packages.sig
$

```

Sensors automatically check for updates in the repository every 5 minutes. Once new files are in place, the update will happen on the next cycle.

