Intel® RSP SW Toolkit - Gateway

Installation & User's Guide

Document Number: 338443-001
Document Revision: 2018.12.13

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.

Revision History

Revision	Description
2018.11.14	Initial draft.
2018.12.11	Updated screenshots.
2018.12.13	Branding updates.

Table of Contents

1 INTRODUCTION 6

1.1 TERMINOLOGY 6
1.2 REFERENCE DOCUMENTS 6

2 SENSOR PRODUCT DESCRIPTION 7

3 SYSTEM DESCRIPTION 8

3.1 DATA FLOW 8
3.2 APPLICATIONS 9
3.3 GATEWAY 9
3.4 INTEL® RFID SENSOR PLATFORM (INTEL® RSP) 9
3.5 JSON RPC 10
3.5.1 Request Object 10
3.5.2 Notification Object 10
3.5.3 Response Object 11
3.5.4 Error Codes 11
3.6 MQTT 12
3.7 REST 12
3.8 SYSTEM REQUIREMENTS 13
3.8.1 Hardware Requirements 13
3.8.2 Software Requirements 13
3.8.3 Network Requirements 13

4 CONCEPT OF OPERATION 14

4.1 CONFIGURATION: SENSOR FACILITY 14
4.2 CONFIGURATION: SENSOR PERSONALITY 14
4.2.1 EXIT Personality 14
4.2.2 Point of Sale (POS) Personality 15
4.2.3 FITTING_ROOM Personality 15
4.3 RFID BEHAVIORS 15
4.4 SCHEDULING TAG READ ACTIVITY 15
4.4.1 INACTIVE 16
4.4.2 ALL_ON 16
4.4.3 ALL_SEQUENCED 16
4.4.4 FROM_CONFIG 16
4.4.5 Example Schedule Configuration 17
4.5 PROCESSING TAG READS 18
4.5.1 Tag States 18
4.5.2 Tag Events 18

4.6 Example Behavior 20
4.7 Tag Location Algorithms 21

5 INTEL® RSP SW TOOLKIT - GATEWAY 23

5.1 Build and Deploy 23
5.2 Project Runtime Folders 24
5.3 Configure 25
5.4 Certificate Generation 28
5.5 Run 28
5.6 Sensor Provisioning Tokens 29
5.6.1 Discovery and Mutual Authentication Process 30

6 COMMAND LINE INTERFACE (CLI) 31

6.1 Quit 31
6.2 Upstream 31
6.3 Downstream 31
6.4 Log 31
6.5 Inventory 32
6.6 Scheduler 33
6.7 System 33
6.8 Tokens 33
6.9 Version 34
6.10 Sensor 34

7 UPSTREAM DATA CHANNEL API 36

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

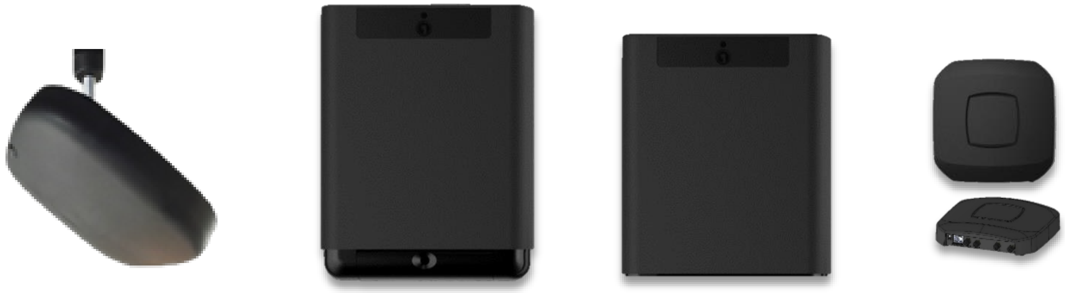
Term	**Description**
RSP	RFID Sensor Platform
NFC	Near Field Communications

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.



RSP-9000 RSP-H4000 RSP-H3000 RSP-H1000

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.

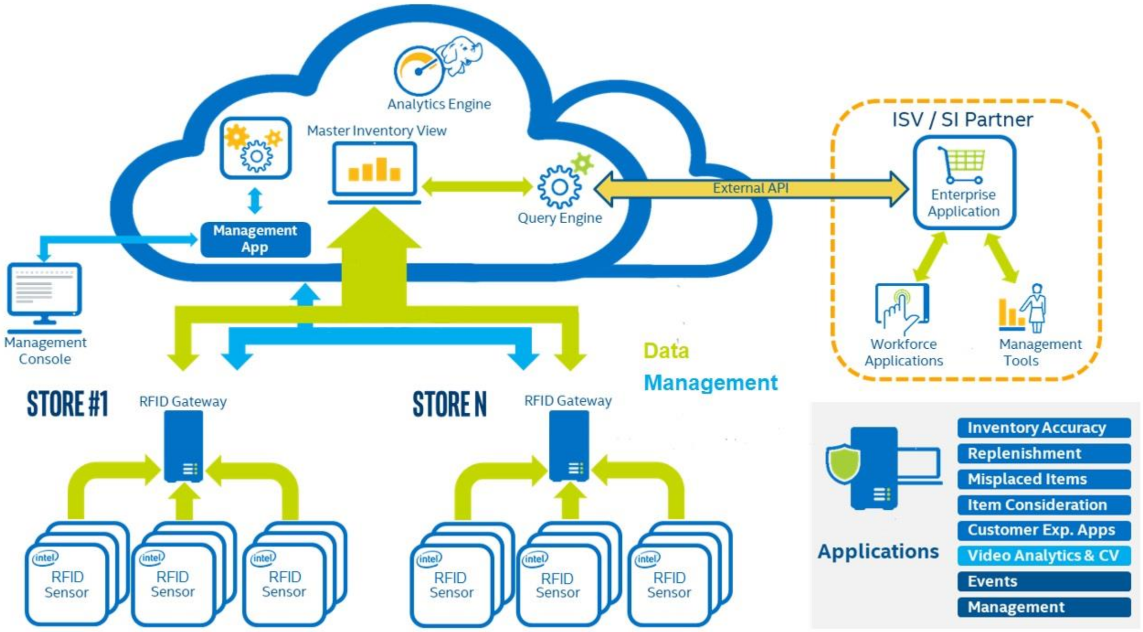


Figure 2: Example System Data Flow

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 on 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.

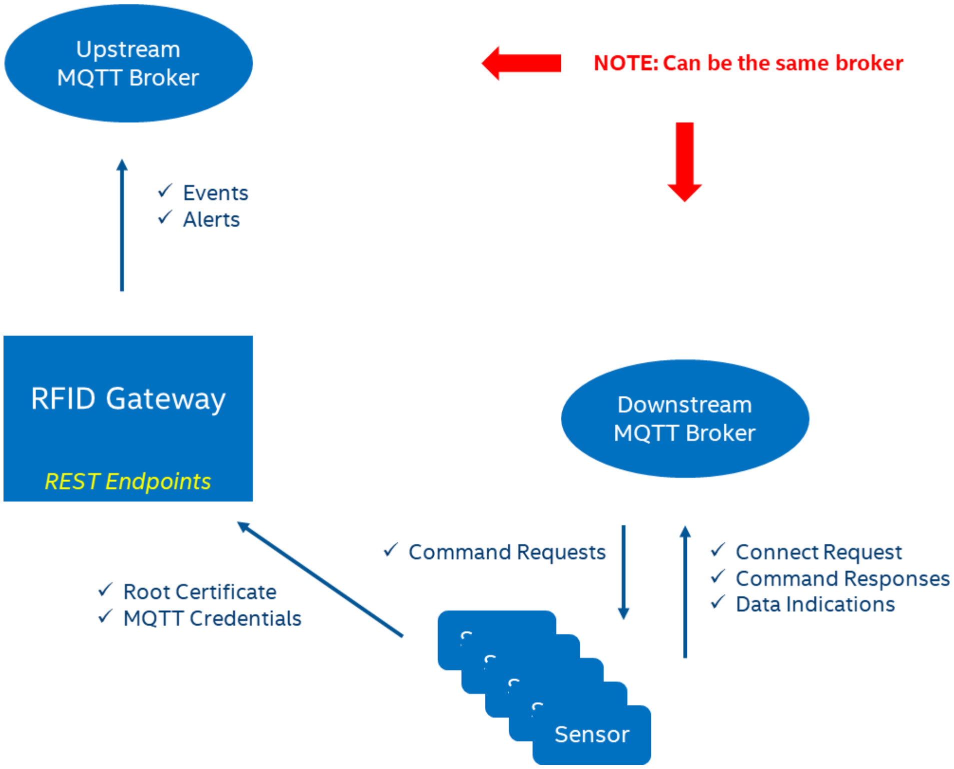


Figure 3: Simple Data Flow Architecture

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	tcp	http, https for software management
22, 62939	ssh	Gateway and Sensor shell connectivity for trouble shooting and log viewing

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 raw 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
- 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. Currently, there are two unique personalities; EXIT and POS.

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 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.

4.4 Scheduling Tag Read Activity

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 XXX 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.

4.4.1 INACTIVE

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

4.4.2 ALL_ON

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

4.4.3 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.

4.4.4 FROM_CONFIG

Schedule Configurations are the best way to manage a population of Sensors. Several configuration topics are all combined into one text file and it 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 just a collection of one or more sensors that will be managed as a group. A Cluster contains one or more Sensor Groups and has an associated Facility, Behavior, and optional Personality.

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.

4.4.5 Example Schedule Configuration

The following JSON is an example of a schedule configuration:

```json
{
  "id": "SampleScheduleConfiguration",
  "clusters": [
    {
      "facility_id": "SalesFloor01",
      "behavior_id": "DefaultExit",
      "personality": "EXIT",
      "sensor_groups": [
        ["RSP-E00100", "RSP-E00101"]
      ]
    },
    {
      "facility_id": "SalesFloor01",
      "behavior_id": "DefaultFittingRoom",
      "personality": "FITTING_ROOM",
      "sensor_groups": [
        ["RSP-F00200", "RSP-F00201", "RSP-F00202"]
      ]
    },
    {
      "facility_id": "SalesFloor01",
      "behavior_id": "DefaultMobility",
      "sensor_groups": [
        ["RSP-000001", "RSP-000004", "RSP-000007"],
        ["RSP-000002", "RSP-000005", "RSP-000008"],
        ["RSP-000003", "RSP-000006", "RSP-000009"]
      ]
    },
    {
      "facility_id": "BackStock01",
      "behavior_id": "DefaultDeepScan",
      "sensor_groups": [
        ["RSP-000011"],
        ["RSP-000012"],
        ["RSP-000013"]
      ]
    }
  ]
}
```

4.5 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.

4.5.1 Tag States

There are five possible states for a tag.

- *unkown* - 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 and 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.

4.5.2 Tag Events

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

- *arrival* - Indicates tag is "new".
- *moved* - Indicates a tag has changed location from one Sensor area to another.
- *departed* - Indicates a tag has "left"
- *returned* - Indicates a tag that was previously departed has shown up again..

The figure and associated table below shows how the RFID tag state transitions occur based on the various RFID tag read conditions and identifies when the various events are sent on the Upstream MQTT connection.

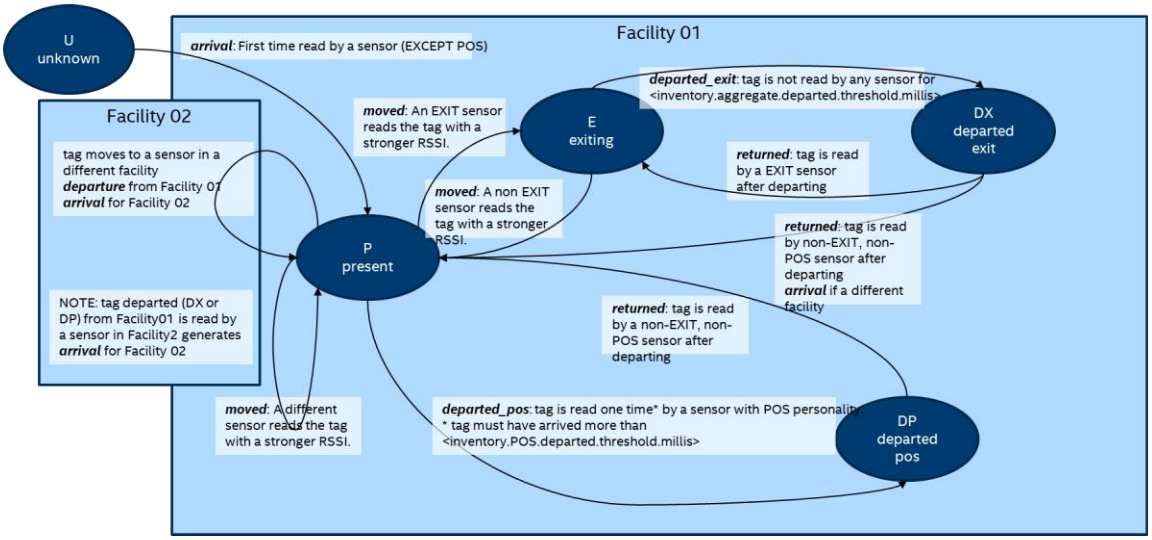

U
unknown
Facility 01
arrival: First time read by a sensor (EXCEPT POS)
Facility 02
tag moves to a sensor in a different facility
departure from Facility 01
arrival for Facility 02
NOTE: tag departed (DX or DP) from Facility01 is read by a sensor in Facility2 generates arrival for Facility 02
moved: An EXIT sensor reads the tag with a stronger RSSI.
E
exiting
departed_exit: tag is not read by any sensor for <inventory.aggregate.departed.threshold.millis>
DX
departed
exit
returned: tag is read by a EXIT sensor after departing
moved: A non EXIT sensor reads the tag with a stronger RSSI.
P
present
returned: tag is read by non-EXIT, non-POS sensor after departing
arrival if a different facility
returned: tag is read by a non-EXIT, non-POS sensor after departing
moved: A different sensor reads the tag with a stronger RSSI.
departed_pos: tag is read one time* by a sensor with POS personality
* tag must have arrived more than <inventory.POS.departed.threshold.millis>
DP
departed
pos

Figure 4 Tag State Diagram

Table 3 Tag State Table

previous	current	event	previous location	current location	personality	condition
U	P	arrival	(unknown)	F1-RSP01-0	none	Tag is first seen by a non-POS sensor
U	U	---	(unknown)	F1-RSP-P-0	POS	Tag is first seen by a POS sensor
P	P	moved	F1-RSP01-0	F1-RSP01-1	none	Tag rssi is stronger on a different antenna port
P	P	moved	F1-RSP01-1	F1-RSP02-0	none	Tag rssi is stronger on different sensor in the same facility
P	P	departed / arrival	F1-RSP02-0	F2-RSP03-0	none	Tag rrsi is stronger beneath a different sensor in a different facility
P	E	---	F2-RSP03-0	F2-RSP04-0	EXIT	Tag is read a single time by exit sensor AND scheduler is not DEEP_SCAN
E	E	moved	F2-RSP03-0	F2-RSP04-0	EXIT	Tag rssi is stronger on exit sensor
E	P	moved	F2-RSP04-0	F2-RSP03-0	none	Tag rssi is stronger on non-exit sensor in same facility
E	P	departed / arrival	F2-RSP04-0	F1-RSP01-0	none	Tag rrsi is stronger on non-exit sensor in a different facility
E	DX	departed	n/a	n/a	n/a	Tag has not been read by any sensor for DT1*
DX	E	returned		F2-RSP-E-0	EXIT	Tag is read by exit sensor after it has departed previously
DX	P	returned		F2-RSP03-0	none	Tag is read by non-exit sensor after it has departed previously
DX	DX	---	F2-RSP-E-0		POS	Departed Tag is read by POS
P	DP	departed	n/a	F2-RSP-P-0	POS	Tag is read ONCE by pos sensor as long as the tag arrival is after DT2*
DP	DP	---	F2-RSP-P-0	F2-RSP-P-0	POS	Tag is read by POS sensor
DP	P	returned	F2-RSP-P-0	F2-RSP03-0	none	Tag is read after DT3* has elapsed
DP	P	arrival	F2-RSP-P-0	F1-RSP01-0	none	Tag is read in different facility after DT3* has elapsed

4.6 Example Behavior

Here is an example of the contents of a behavior.

```
{
  "id": "AutoMobility-S1",
  "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
}
```

All of the behavior files are located in the directory “<DEPLOY_DIRECTORY>/config/behaviors”. Specific naming criteria is used to differentiate some of the default behaviors.

Shown below is an example list of behaviors.

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

4.7 Tag Location Algorithms

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 RSSI from the new Sensor must be 'T' dBm better than the RSSI of the previous location. RSSI values are time-averaged using weights. A “mobility profile” defines the parameters of the RSSI slope formula used in the tag movement decay algorithm and custom profiles can be configured 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; }
```

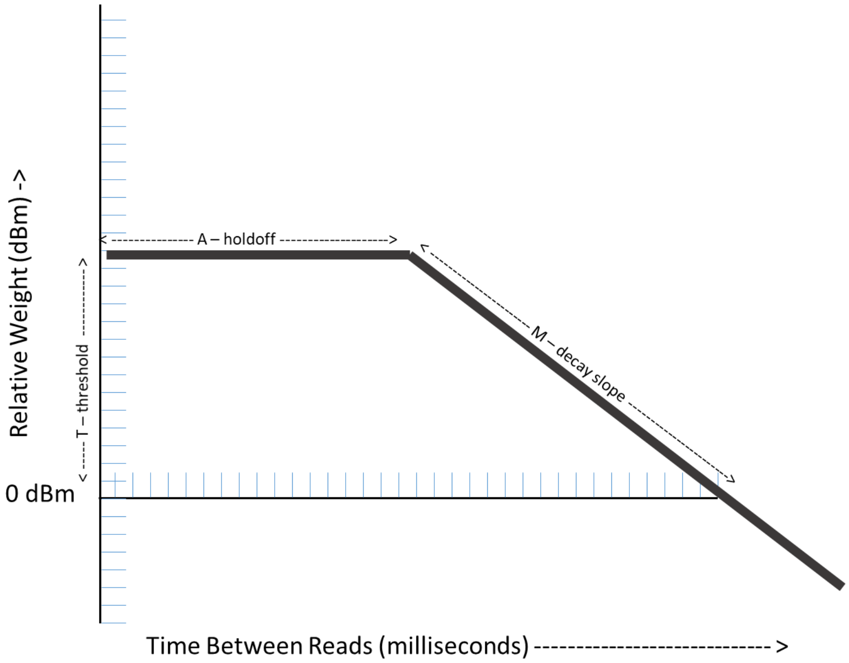


Figure 5 Mobility Profile

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
}
```

5 Intel® RSP SW Toolkit - Gateway

The source code and more information regarding the Gateway can be downloaded from the Intel® Open Source Portal https://01.org. The project is located in the "Developer Toolkits" section under "Intel® RSP SW Toolkit". Follow the "GIT REPO" link to obtain the software.

5.1 Build and Deploy

These instructions assume a clean Ubuntu 18.04 LTS installation. There are also references to PROJECT_DIR and DEPLOY_DIR to indicate directories of your choice.

```
#-- install development tools
sudo apt-get install openjdk-8-jdk git gradle

#-- install runtime packages
sudo apt-get install mosquitto avahi-daemon ntp ssh gradle

#-- clone the project
cd $PROJECT_DIR

# TODO: update with 01.org reference
git clone https://github.com/intel/rsp-sw-toolkit.git

#-- build an archive suitable for deployment
cd gateway
gradle buildTar

#-- Deploy the project
cd $DEPLOY_DIR
tar -xf ${PROJECT_DIR}/build/distributions/gateway-1.0.tgz
```

5.2 Project Runtime Folders

- 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
- results
- snapshot - Location for inventory snapshots.
- stats - Location for tag and Sensor statistics
- tagread - Location for raw tag read files (when enabled)

5.3 Configure

Set the application parameters by editing <HOME_DIR>/config/gateway.cfg”.

```
#-------------------------------------------------------------------------
#-- 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
#-------------------------------------------------------------------------
```

5.4 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 this 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.

```
#--
#-- Generate certificates and keys if needed.
#--
cd <DEPLOY_DIR>/gateway
mkdir -p ${DEPLOY_DIR}/gateway/cache
cd ${DEPLOY_DIR}/gateway/cache
${DEPLOY_DIR}/gateway/gen_keys.sh
```

5.5 Run

```
#--
#-- Start the gateway application.
#--
<DEPLOY_DIR>/gateway/run.sh
```

5.6 Sensor Provisioning Tokens

The Gateway supports a mutual authentication scheme with Sensors as shown in the figure below. 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, two pieces of information need to be stored on the Sensor using the "RSP NFC Provisioning App" prior to connecting to the Gateway; they are a hash of the root CA certificate and a 64 character security token. The certificate hash is used by the Sensor to verify the root CA certificate it downloads as part of the connection sequence. This is the <DEPLOY_DIR>/cache/ca.crt generated in the previous section. The security token is sent from the Sensor to the Gateway in the connect request. The Gateway confirms that the token is still valid or does not allow the Sensor to connect.

Tokens can be generated using the follwing sequence:gateway command line interface;

```
rfid-gw> tokens generate
-------------------------------------------------------------------------------------------
provision token exported to: <DEPLOY_DIR>/gateway/exported-tokens/token_20181130_135713.json
{
  "username" : "tshockley",
  "token" : "2BD3C55CAE6E1E4113CC695A9C486D8A588B00A690277A2E2BECC0B111C4DEBC",
  "generatedTimestamp" : 1543611433883,
  "expirationTimestamp" : 1546203433883
}
-------------------------------------------------------------------------------------------
```

By default, a token is only valid for 24 hours, but this can be changed when generating tokens. When new tokens are generated, they are exported to the directory <DEPLOY_DIR>/ace-point-gateway/exported-tokens with a filename that includes the local timestamp of when they were generated.

The generated token(s) and root CA certificate are downloaded to the RSP NFC Provisioning Application as described in section TBD of the NFC application's User Guide. Both items are programmed into the Sensor via an NFC capable Android device. See the "Intel® RSP SW Toolkit – Sensor NFC App Installation & User Guide" for detailed instructions on programming the Sensor.

5.6.1 Discovery and Mutual Authentication Process

The Gateway Discovery and Mutual Authentication process begins by provisioning the Sensor via NFC with a Token/Hash value pair. Once power-on and boot is complete, the Sensor uses the information contained in the JmDNS Broadcast to synchronize time and download the Root Certificate. If a Sensor has been provisioned with an NFC tag, the sha256 hash of this certificate must match the Hash in the Token/Hash value pair. While retrieving the MQTT credentials, the Sensor provides the Gateway with the Token from the Token/Hash value pair. If the Token is known to the Gateway and still valid, the Gateway will trust the Sensor and provide the credentials necessary to connect to MQTT and thereby connect to the Gateway.

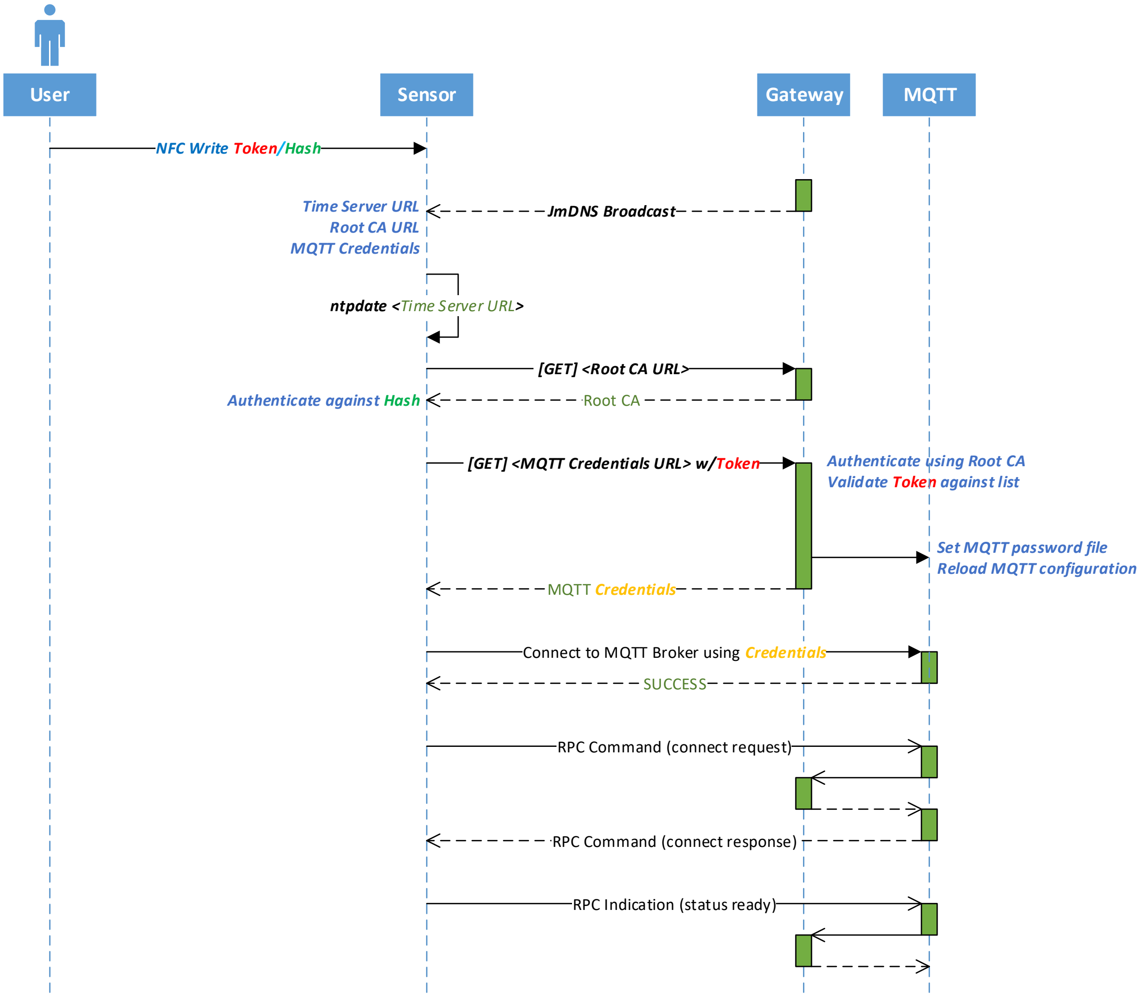

User
Sensor
Gateway
MQTT
NFC Write Token/Hash
Time Server URL
Root CA URL
MQTT Credentials
JmDNS Broadcast
ntpdate <Time Server URL>
[GET] <Root CA URL>
Authenticate against Hash
Root CA
[GET] <MQTT Credentials URL> w/Token
Authenticate using Root CA
Validate Token against list
Set MQTT password file
Reload MQTT configuration
MQTT Credentials
Connect to MQTT Broker using Credentials
SUCCESS
RPC Command (connect request)
RPC Command (connect response)
RPC Indication (status ready)

Figure 6 Discovery and Mutual Authentication Process

6 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. By 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 about what the available options are from 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>
quit         upstream     downstream   log          inventory    scheduler
system       tokens       version      sensor
```

6.1 Quit

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

6.2 Upstream

```
--------------------------------------------------------------------------------------------
USAGE

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

6.3 Downstream

```
--------------------------------------------------------------------------------------------
USAGE

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

6.4 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
---------------------------------------------------------------------------------------------
```

6.5 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 !! 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
```

```
--------------------------------------------------------------------------------------
```

6.6 Scheduler

```
--------------------------------------------------------------------------------------------
USAGE

> 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.file <file_path>
    Load a schedule from the specified JSON file path

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

6.7 System

```
--------------------------------------------------------------------------------------------
USAGE

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

6.8 Tokens

```
--------------------------------------------------------------------------------------------
USAGE

> tokens show
    Show existing provisioning tokens

> tokens show.duration
    Show the valid duration used when generating new provisioning tokens

> tokens set.duration.infinite
    Set the valid duration of new provisioning tokens to never expire

> tokens set.duration.days <number>
    Set the valid duration of new provisioning tokens in days (must be > 0)

> tokens set.duration.hours <number>
    Set the valid duration of new provisioning tokens in hours (must be > 0)

> tokens generate
    Generate a new provisioning token

> tokens export
    Exports all tokens to /home/tshockley/Test/ace-point-gateway/exported-tokens

> tokens delete <token>
    Delete an existing provisioning token
    WARNING: TOKEN IS NOT RECOVERABLE. Use with caution
--------------------------------------------------------------------------------------------
```

6.9 Version

```
--------------------------------------------------------------------------------------------
USAGE

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

6.10 Sensor

```
--------------------------------------------------------------------------------------------
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>...
    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 (faciility, 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
--------------------------------------------------------------------------------------------
```

7 Upstream Data Channel API

The Gateway generates JSON RPC Notifications on the Upstream MQTT broker for inventory events and device alerts. A description of these notifications is provided in the table below.

Table 4 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