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, Ostro, 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 © 2017, Intel Corporation. All rights reserved.
## Contents

1.0 Introduction .................................................................................................................. 6
  1.1 Terminology ................................................................................................................ 6
  1.2 Reference Documents ............................................................................................... 7
  1.3 Customer Support ..................................................................................................... 7

2.0 Intel® SHDAP SDK Services and Privileges ................................................................. 8
  2.1 Application Development Framework ...................................................................... 9
  2.2 Connectivity and Features Supported by SDK ....................................................... 9

3.0 Rule Engine .................................................................................................................. 11
  3.1 Grammar .................................................................................................................. 11
    3.1.1 Rules ................................................................................................................ 11
    3.1.2 Sample Rules ................................................................................................... 12
  3.2 Limitations .............................................................................................................. 13

4.0 Intel® SHDAP SDK Services and Privileges ................................................................. 15
  4.1 ROOT Privileges ...................................................................................................... 15
  4.2 Intel® SHDAP Privileged User Privileges ............................................................... 15
  4.3 Intel® SHDAP Services ........................................................................................... 16
  4.4 Guidelines to Enable New Services Along with Intel® SHDAP Services .............. 16
  4.5 Watchdog Timer ...................................................................................................... 17
  4.6 Gateway On-Boarding ............................................................................................. 17
  4.7 Intel® SHDAP Recovery Mechanism ....................................................................... 18

5.0 Accessing SDK Logs ................................................................................................... 20

6.0 Developing Smart Home Applications ........................................................................ 22
  6.1 Intel® SHDAP API Structure .................................................................................... 22
  6.2 Multiple Application Feature .................................................................................. 25
  6.3 Group Management ................................................................................................. 25

7.0 Intel® SHDAP SDK Sample Applications ................................................................... 26
  7.1 HTML5 Sample Application .................................................................................... 26
    7.1.1 Application Features ....................................................................................... 26
    7.1.2 Making the Application as a Service ............................................................. 28
    7.1.3 Enhancing the Application .......................................................................... 29
  7.2 Node.js*-Based Sample Applications ..................................................................... 29
    7.2.1 Connectivity Sample Application ................................................................ 29
    7.2.2 Rule Engine Sample Application ................................................................. 30
  7.3 Intel® XDK Sample Applications ........................................................................... 31
  7.4 Cloud Connector Sample Applications ................................................................... 31
    7.4.1 Amazon Web Services* (AWS*) Sample Application .................................. 32
    7.4.2 Microsoft* Azure* Sample Application ....................................................... 36
7.5 Adding a New Smart Home Application ................................................................. 40

8.0 Configuring Philips® Hue Bridge Plugin ...................................................................... 42
8.1 Configuring Philips® Hue Bridge ................................................................................ 42
8.2 Updating Bridge Details ........................................................................................... 44

9.0 Supported Device Configurations ........................................................................... 45
9.1 ZigBee® Device Configuration .................................................................................. 45
9.1.1 Using MMB Sample Application ......................................................................... 47
9.2 Z-Wave® Device Configuration ................................................................................. 48
9.2.1 Adding Support for a New Vendor Z-Wave® to SDK ............................................. 51
9.2.2 Configuring Plugin to Map Vendor-Specific Data to Z-Ware Specified Information ........................................................................................................ 53
9.2.3 Configuring the Z-Wave Device Parameter ......................................................... 54

10.0 Supported Devices ................................................................................................. 57

11.0 Troubleshooting Guide .......................................................................................... 67
11.1 Z-Wave® Stack ........................................................................................................ 67
11.1.1 Using Z-Wave® Sample Application .................................................................. 67
11.2 Verify Bluetooth Interface on Intel® SHDAP Gateways .......................................... 68

Figures

Figure 1. Gateway Applications Communicating with Smart Home Devices through Intel® Smart Home Development Platform (Intel® SHDAP) ........................................................................... 8
Figure 2. Intel® SHDAP ................................................................................................. 27

Tables

Table 1. Terminology ....................................................................................................... 6
Table 2. Reference Documents ...................................................................................... 7
Table 3. Log Levels and Descriptions ........................................................................... 20
Table 4. List of Supported Clusters ............................................................................... 45
Table 5. Properties Supported in Implemented Clusters ............................................... 45
Table 6. Commands for the Sample Application ............................................................. 47
Table 7. List of Supported Command Classes ............................................................... 48
Table 8. Command Class Names and Supported Properties ........................................ 49
Table 9. OIC Device Type Mapping .............................................................................. 52
Table 10. List of Devices Tested in This Release .............................................................. 57
Table 11. Meanings of Value1 and Value2 for Different Devices ..................................... 65
## Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>March, 2017</td>
<td>2.0</td>
<td>Updated for Release 1.2</td>
</tr>
<tr>
<td>February, 2017</td>
<td>1.9</td>
<td>Modified the link for IPS</td>
</tr>
<tr>
<td>January 2017</td>
<td>1.8</td>
<td>Updated for Release 1.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added Section 11.2</td>
</tr>
<tr>
<td>December 2016</td>
<td>1.7</td>
<td>Updated for Release 1.0.1</td>
</tr>
<tr>
<td>November 2016</td>
<td>1.6</td>
<td>Updated for Release 1.0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added Sections 7.1.2, 7.4.1.1, 7.4.1.2, 7.4.1.3, 7.4.2.1, 7.4.2.2, and 9.2.3.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated Section 11.1.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Removed &quot;Using PYZIP Utility&quot;</td>
</tr>
<tr>
<td>October 2016</td>
<td>1.5</td>
<td>Updated for Pre-Release 0.82</td>
</tr>
<tr>
<td>August 2016</td>
<td>1.4</td>
<td>Updated for Pre-Release 0.81</td>
</tr>
<tr>
<td>July 2016</td>
<td>1.3</td>
<td>Updated for Pre-Release 0.8</td>
</tr>
<tr>
<td>May 2016</td>
<td>1.2</td>
<td>Updated for Pre-Release 0.6</td>
</tr>
<tr>
<td>May 2016</td>
<td>1.1</td>
<td>Updated for Pre-Release 0.51</td>
</tr>
<tr>
<td>April 2016</td>
<td>1.0</td>
<td>Intel® SHDAP Developer Guide Pre Release 0.5</td>
</tr>
</tbody>
</table>
1.0 Introduction

This Developer Guide contains the following areas for the usage of Intel® Smart Home Development Acceleration Platform (Intel® SHDAP) Software Development Kit (SDK):

- The SDK framework
- Rule engine grammar
- Using the SDK APIs to develop smart home applications
- Philips* Hue bulb plugin configuration
- Supported devices

1.1 Terminology

Table 1. Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP</td>
<td>Access Point</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>APK</td>
<td>Android* Application Package File</td>
</tr>
<tr>
<td>DHCP</td>
<td>Dynamic Host Configuration Protocol</td>
</tr>
<tr>
<td>Gateway</td>
<td>Controller to control the end devices/smart things</td>
</tr>
<tr>
<td>GW</td>
<td>Gateway</td>
</tr>
<tr>
<td>IAM</td>
<td>Identity and Access Management</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>Intel® SHDAP</td>
<td>Intel® Smart Home Development Acceleration Platform</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>OIC</td>
<td>Open Interconnect Consortium</td>
</tr>
<tr>
<td>Ostro™ OS</td>
<td>Yocto Project*-based metadata and source code for IoT devices</td>
</tr>
<tr>
<td></td>
<td><a href="https://ostroproject.org/">https://ostroproject.org/</a></td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Socket Shell</td>
</tr>
<tr>
<td>Things / End Devices</td>
<td>General term for smart home sensors and actuators</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
</tbody>
</table>
1.2 Reference Documents

Table 2. Reference Documents

<table>
<thead>
<tr>
<th>Document</th>
<th>Document No./Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Smart Home Development Acceleration Platform (Intel® SHDAP)</td>
<td>567375</td>
</tr>
<tr>
<td>Software Development Kit (SDK) Quick Start Guide</td>
<td></td>
</tr>
<tr>
<td>Intel® Smart Home Development Acceleration Platform (Intel® SHDAP)</td>
<td>567376</td>
</tr>
<tr>
<td>Software Development Kit (SDK) Developer Guide and Reference Document</td>
<td></td>
</tr>
<tr>
<td>Intel® Smart Home Development Acceleration Platform (Intel® SHDAP)</td>
<td>567377</td>
</tr>
<tr>
<td>Software Development Kit (SDK) SDK API Specification</td>
<td></td>
</tr>
<tr>
<td>Intel® Smart Home Development Acceleration Platform (Intel® SHDAP)</td>
<td>567465</td>
</tr>
<tr>
<td>Software Development Kit (SDK) Z-Wave Interface Guide</td>
<td></td>
</tr>
<tr>
<td>Getting Started with the Intel® XDK</td>
<td>567736</td>
</tr>
<tr>
<td>Intel® Smart Home Development Acceleration Platform (Intel® SHDAP)</td>
<td>567463</td>
</tr>
<tr>
<td>Software Development Kit (SDK) Dependency Modules</td>
<td></td>
</tr>
<tr>
<td>Intel® Smart Home Development Acceleration Platform (Intel® SHDAP)</td>
<td>567737</td>
</tr>
<tr>
<td>Software Development Kit (SDK) Event Bus Guide</td>
<td></td>
</tr>
<tr>
<td>Intel® Smart Home Development Acceleration Platform (Intel® SHDAP)</td>
<td>571011</td>
</tr>
<tr>
<td>Production Image Build Guide</td>
<td></td>
</tr>
</tbody>
</table>

1.3 Customer Support

Contact Intel using the Intel® Premier Support tool for technical support, including answers to questions not addressed in this document. Contact your Intel representative to set up the Intel Premier Support account.

§
2.0 **Intel Smart Home Development Platform (Intel® SHDAP) SDK**

Intel® Smart Home Development Platform (Intel® SHDAP) is an open horizontal platform that is flexible and easy to use in creating a smart home "hub." Intel SHDAP provides a software framework that makes it easy for devices and applications to discover and communicate with each other. Intel SHDAP also supports a rule engine, which enables you to create rules based on different needs, configure necessary actions, and inject the actions into the Gateway at run time. You can write applications for device interoperability regardless of transport layer and manufacturer of smart home end devices.

**Figure 1.** Gateway Applications Communicating with Smart Home Devices through Intel® Smart Home Development Platform (Intel® SHDAP)
The Intel SHDAP core consists of two major runtime components:

- Things Abstraction Layer (TAL): Provides secure connectivity and communication with smart home devices.
- API Runtime: Provides rule engine execution functionality.

These runtime components are loosely coupled through the Intel SHDAP message bus and execute in parallel as Ostro™ OS processes to provide the best performance.

The TAL is developed in compliance with OIC specifications, and therefore can communicate with OIC devices (smart things) seamlessly. To support non-OIC devices, TAL uses Protocol Plugin Manager (PPM), which performs the necessary translation or acts as a bridge between OIC and device-specific connectivity and protocols.

## 2.1 Application Development Framework

The Application Development Framework contains the Node.js* static library that provides APIs to communicate with smart home end devices, and to manage rules, debugging, device management, and configuration. Intel SHDAP APIs work in asynchronous mode, which means the results for API calls are provided as Node.js events to the application. You can make calls to these APIs to perform the required functionality and read the result on events.

The Framework provides Node.js static library interfaces that enable you to make the call to perform the functionality as mentioned in the API specification. Sample applications can be used as a quick start reference while developing innovative use cases and end user applications.

Intel SHDAP provides simple but powerful and secure programming interfaces to help you develop innovative smart home applications and use cases. The Intel SHDAP SDK provides support for a wide range of device manufacturers, device types, and connectivity.

Intel SHDAP also includes an integrated development environment to help you fast-track development and for better code management.

## 2.2 Connectivity and Features Supported by SDK

The following features and connectivity options are supported in this release:

- Connectivity with Z-Wave* Things
- Connectivity with ZigBee* Things
- Connectivity with Bluetooth* low energy (LE) Things
- Gateway and device onboarding over Wi-Fi
- Philips* Hue bridge connection over LAN
- SDK APIs to discover, onboard, read, and write Smart Things (end devices) seamlessly.
- Rule Engine programming APIs
- Intel SHDAP SDK Security features
- Debugging features
- Smart Home Things (Devices) and services represented as per OIC specifications
3.0 Rule Engine

The rule engine component enables you to define regular activities or policies as rules using PEG.js grammar. After defining the rule, the rule engine evaluates and executes the rule in real time.

3.1 Grammar

The rules, which are written in English-like statements, are based on PEG.js grammar (http://pegjs.org). See a sample rule in Section 3.1.1.

PEG.js grammar is available at /home/shdap/Intel/shdap_sdk/tools/rulelang.pegjs

3.1.1 Rules

The following is a sample rule template:

```
ruleset: <ruleset name>
appid: <app id>
rule: <rule name>
{
when
<triggers>
if( <condition> )
then
<actions>
else
<actions>
end
}
```

There are two types of triggers:
- timer
- state change

There are two types of actions:
- change_state
- notify
3.1.2 Sample Rules

3.1.2.1 Simple Rules

State change trigger-based rule with Device ID:

```
rule: Bedroom_Rule

{ when
  bedroom_door_trigger: "lockState" of "oic:000d6f0002fb71c6"
  changed from "Locked" to "Unlocked"
  if( value of "oic:001788ffee129469" == "1" )
  then
    @change_state("oic:001788ffee129469", {"value" : "0"})
  end
}
```

State change trigger-based rule with Alias:

```
rule: Bedroom_Rule

{ when
  bedroom_door_trigger: "lockState" of "bed_room_door"
  changed from "Locked" to "Unlocked"
  if( value of "HueBulb" == "1" )
  then
    @change_state("HueBulb", {"value" : "0"})
  end
}
```

Timer-based rule with Device ID:

```
rule: timer_YaleDoorLock_rule

{ when
  night: timer repeat "18:35:00"
  if( 1 == 1 )
  then
    @change_state("oic: 000d6f0002fb71c6", {"lockState" : "Locked"})
  end
}
```
3.1.2.2 Complex Rules

Sample complex rule with Alias:

```plaintext
ruleset: smartHomeDap_basic_rules_PhilipsHueBulb
appid: sample_app
rule: complex_PhilipsHueBulb_rule
{
   when
   night: timer repeat "19:35:00"
   hall_PhilipsHueBulb_trigger: "hue" of "myHueBulb" between 20 and 40
   if( "lockState" of "doorLock" != "Unlocked" )
   then
      @change_state("myHueBulb", { "hue" : 120, "saturation" : 254})
      @notify("hue of myHuebulb increased")
   else
      @change_state("myHueBulb", { "hue" : 60, "saturation" : 254})
      @notify("hue of myHuebulb decreased")
   end
}
```

3.2 Limitations

The Intel SHDAP rule engine supports rule creation using the device alias name. The following are guidelines for using device alias name in the rules:

- If a rule with a device alias is added to the rule engine, the rule engine makes a real-time check in the device registry to ensure that the device with the alias name is present in the system.

- If the device is removed, the rule associated with the device stays in the system and must be explicitly deleted using the deleteRule API.
The rule engine supports rules that are written using both DeviceId and DeviceAlias. However, Intel recommends creating rules using DeviceAlias instead of DeviceId.
4.0 Intel® SHDAP SDK Services and Privileges

The Intel SHDAP SDK uses Linux multi-user framework, Access Control List (ACL), and Discretionary Access Control (DAC) security features.

4.1 ROOT Privileges

- The following modules run as root user and root group:
  - Watchdog Timer (WDT)
  - Z/IP Gateway Server
- Use the following commands to Start / Stop the above modules
  - Watchdog Timer (WDT)
    - systemctl start watchdogtimer.service
    - systemctl stop watchdogtimer.service
  - Z/IP Gateway Server
    - systemctl start shdap-root.service
    - systemctl stop shdap-root.service
- Other users are not allowed to run this service (except the root user)

4.2 Intel® SHDAP Privileged User Privileges

- "shdap" is a privileged user, which is created by the installer during installation.
- The following modules run as "shdap" user and "intel" group:
  - Broker
  - TAL
  - PPM
  - API Runtime
- The following command is used to start the preceding modules:
  - $ systemctl start shdap.service
- The following commands are used to stop the preceding modules:
  - $ systemctl stop shdap.service
  - $ systemctl stop api.service
  - $ systemctl stop broker.service
  - $ systemctl stop tal.service
  - $ systemctl stop ppm.service
- The status of services can be checked with the following commands:
  - $ systemctl status shdap.service
  - $ systemctl status api.service
  - $ systemctl status broker.service
4.3 Intel® SHDAP Services

- Modules that need to run as the root user are installed on the /home/root/Intel/ directory. The Gateway onboarding and startup scripts reside in this directory.
- Modules that need to be installed as the shdap user are installed in /home/shdap/Intel/shdap_sdk/<respective_folder>. The Intel SHDAP SDK gets installed there.
- All node module dependent packages are installed in the /usr/lib/node_modules/ directory with the permissions 755 (rwx r-x r-x).
- All third-party libraries (like zmq) are installed in the /usr/lib directory with the permissions 755 (rwx r-x r-x).
- The common log level file is located at /home/shdap/Intel/shdap_sdk/bin/log_level.txt and given full permissions 777.
- Once the SDK installation is complete, log in to the board using the application user name to run the sample applications.
  $ ssh app@<IP address>

4.4 Guidelines to Enable New Services Along with Intel® SHDAP Services

- Copy the service file to the /lib/systemd/system/ directory on the Gateway.
- If a service has to be started during Gateway bootup, follow these steps:
  - The service file should contain the following lines of code:
    
    [Install]
    WantedBy=multi-user.target

  - Execute the command
    $ systemctl enable <name of service file>

- After copying the service to /lib/systemd/system/ or after enabling the service, for these changes to take effect, execute the following command:

  $ systemctl daemon-reload

*Note:* The existing services in Intel SHDAP are enabled by default during installation.
4.5 **Watchdog Timer**

How the watchdog timer helps:

When the CPU utilization is full, system memory is full, or if there is a deadlock situation resulting in the system going into a hung-up state, the watchdog timer triggers a system reboot.

How to configure or enable/disable if required:

- The watchdog timer can be enabled by running the following command with root privileges where `<seconds>` specifies the time to system reboot when system fault is triggered:
  ```
  $ watchdog enable <seconds>
  ```

- The watchdog timer can be disabled by running the following command with root privileges:
  ```
  $ watchdog disable
  ```

- Currently the time-out for the watchdog timer is set to 30 seconds in the `watchdogtimer-start.sh` file located in the directory `/home/root/Intel/startup`.

4.6 **Gateway On-Boarding**

Gateway On-Boarding is an experimental feature, and may not work at times or may be slow.

The following are the steps for Gateway on-boarding:

1. Enter the following command:
   ```
   connmanctl
   ```

2. Wait for a connmanctl prompt to appear:
   ```
   connmanctl>
   ```

3. Enter the command “enable wifi”:
   ```
   connmanctl > enable wifi
   ```

4. Wait for the following message:
   ```
   Enabled wifi
   ```

5. Enter the command “agent on”:
   ```
   connmanctl > agent on
   ```

6. Wait for one of the following two messages:
   ```
   Agent registered
   Agent already registered
   ```

7. Enter the command “scan wifi”: 
connmanctl > scan wifi

8. Wait for the following message:
   Scan completed for wifi

9. Enter the command "quit":
   connmanctl > quit

10. Put the Gateway in tether mode using gwOnboarding.js.
    $ cd /home/shdap/Intel/shdap_sdk/gatewayonboarding
    $ node gwOnboarding.js

11. On a mobile/tablet preferably with Android* Marshmallow OS or later versions (in versions earlier than Android Marshmallow, the device switches from Intel SHDAP to the previous AP), search for an access point with the name “test_SHDAP” and connect to it. Key in the security key “12345678” when prompted.

12. Open a browser. At the address bar, enter “http://192.168.0.1”.

13. From the dropdown list, select the SSID of the Wireless access point on which to onboard the Gateway.

14. Enter the password of the access point and press Submit.

The Gateway re-starts and boots up on-boarded on the user-provided AP.

4.7 Intel® SHDAP Recovery Mechanism

If any Intel SHDAP module crashes, the Intel SHDAP system will be in an unusable state. There is a mechanism to recover the Intel SHDAP system from such a state.

• Design:
  - All Intel SHDAP modules — Broker, TAL, PPM, and apiRuntime — are separate services.
  - The Broker starts first.
  - apiRuntime, TAL, and PPM wait until the Broker has created all the endpoints. After all the end points are created, the other three services start.
  - These four separate services are made interdependent using flags provided by systemctl called as “PartOf” and “Restart”.
  - If any module crashes, for example, the Broker, the “Restart” flag results in the Broker service restarting and the “PartOf” flag results in the remaining three services for TAL, PPM, and apiRuntime restarting.

• The service files of all the modules are located at the following directory:
  /home/shdap/Intel/shdap_sdk/tools.
To know if the Intel SHDAP services have restarted because of the Intel SHDAP Recovery Mechanism:

- The application will be in an unusable state, that is, no API calls from the application reach the broker.
- For whichever module crashes or fails, for example, if TAL crashes, the following logs appear in “journalctl -f”:

```
  tal.service: Main process exited,
  tal.service: Unit entered failed state.
  tal.service: Failed with result '....'
  tal.service: Service hold-off time over, scheduling restart.
  Stopping Board service...
  Starting Board service...
```

- The preceding logs also appear when other modules crash, with the corresponding module name as the prefix.
- The way to notify the application when the services are recovered is still in progress.
- Once the Intel SHDAP services are recovered and once the reconnect is complete, the user application would need a restart.
5.0 Accessing SDK Logs

The log levels supported are as follows:

Table 3. Log Levels and Descriptions

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Turn OFF logging. This is also the default log level.</td>
</tr>
<tr>
<td>2</td>
<td>CRIT log level that logs SDK application initialization status and read/write operation status.</td>
</tr>
<tr>
<td>3</td>
<td>ERROR log level that logs erroneous events.</td>
</tr>
<tr>
<td>6</td>
<td>INFO log level that logs the progress of running the SDK application.</td>
</tr>
<tr>
<td>7</td>
<td>DEBUG log level that logs everything useful for debug.</td>
</tr>
</tbody>
</table>

To update the log level:

- Stop all SDK services.
- Open the log_level.txt file located in the directory `/home/shdap/Intel/shdap_sdk/bin/` and update with the desired log level.
- Restart the services.

To view the Intel SHDAP log, use the following commands on the Gateway's command prompt, using the root login:

- To view Intel SHDAP SDK logs on the console:
  
  ```
  $ journalctl -f
  ```

- To view Z/IP Gateway server logs on the console:
  
  ```
  $ journalctl -u shdap-root -f
  ```

- To view logs of individual modules:
  
  ```
  $ journalctl -u api.service -f
  $ journalctl -u broker.service -f
  $ journalctl -u tal.service -f
  $ journalctl -u ppm.service -f
  ```

- To capture all Intel SHDAP SDK logs in one file, run the following script from the directory `/home/shdap/Intel/shdap_sdk/shdap/tools/`. The file containing the logs will be created at `/home/shdap/Intel/shdap_sdk/logs/`
  
  ```
  $ sh shdap-logs.sh shdap
  ```
To capture module-wise logs of Intel SHDAP SDK modules, run the following script from the directory /home/shdap/Intel/shdap_sdk/tools/. The separate files containing the individual module logs will be created at /home/shdap/Intel/shdap_sdk/logs/

```
$ sh shdap-logs.sh
```
6.0 Developing Smart Home Applications

Intel® SHDAP is designed to provide smart home application developers simple and secure APIs to discover, onboard, and communicate with smart home devices (for example, light bulbs, thermostats, water leak sensors, and door locks) seamlessly without worrying about the underlying connectivity and device protocols.

Intel SHDAP also provides APIs to access the rule engine in performing home automation. An application developer can set the rules in normal English and the heavy lifting (for example, taking the decision, executing the rules, and performing) is managed by Intel SHDAP runtimes.

Intel SHDAP APIs are created in the Node.js* environment. These APIs are in a separate module and are asynchronous by design. The API call returns immediately to the application and the result of the call is notified through an event to the application. The application registers for the desired callback events and the Intel SHDAP SDK will notify the application whenever these callback events occur.

Intel SHDAP APIs are broadly categorized as follows:

- **Connectivity Access**: APIs to initiate the scan on selected connectivity (for example, ZigBee*, Bluetooth* LE, and Z-Wave* networks) and add or remove the discovered devices.
- **Device Access**: APIs to access the device and read/write its attributes.
- **Rule Engine**: APIs to access the rule engine.
- **Group**: APIs to group the devices.
- **Version**: API-related SDK version information.
- **App init**: API-related SDK initialization information.

6.1 Intel® SHDAP API Structure

Intel SHDAP APIs have the following namespace:

```
intel.smarthome.accelerationplatform
```

The APIs are classified under the following sub-namespaces:

- connection
- deviceaccess
- rule
- log
- version
- group
• appinit

“apievent” is the main event emitter name in Intel SHDAP SDK and is available to application developers under the namespace intel.smarthome.accelerationplatform.

The following code snippet depicts the module and namespace:

```javascript
module.exports = {
  intel: {
    smarthome: {
      accelerationplatform: {
        connection: {
          initiateScan: initiateScan,
          addDevice: addDevice,
          addDeviceByAlias: addDeviceBy Alias,
          removeDevice: removeDevice,
          removeDeviceByAlias: removeDeviceByAlias
        },
        deviceaccess: {
          readDeviceAttribute: readDeviceAttribute,
          readDeviceAttributeByAlias: readDeviceAttributeByAlias,
          writeDeviceAttribute: writeDeviceAttribute,
          writeDeviceAttributeByAlias: writeDeviceAttributeByAlias,
          findDeviceById: findDeviceById,
          findDeviceByAlias: findDeviceByAlias,
          findDevices: findDevices,
          setDeviceAlias: setDeviceAlias,
          updateDeviceAlias: updateDeviceAlias
        },
        rule: {
          addRule: addRule,
          modifyRule: modifyRule,
          deleteRule: deleteRule,
          deleteAllRules: deleteAllRules,
          listAllRules: listAllRules,
          enableRule: enableRule,
          disableRule: disableRule
        },
        log: {
          setModuleLogLevel: setModuleLogLevel
        }
      }
    }
  }
};
```
To access the Intel SHDAP SDK API, Intel SHDAP application developers need the smartHomeDapApi in their application.

The following code snippet describes the usage of Intel SHDAP API:

```javascript
"use strict";
var shdap = require("smartHomeDapApi").shdap; //require the Intel® SHDAP sdk API
/*register for evtResultInitiateScan event on Intel® SHDAP
event emitter 'apievent' */
shdap.apievent.on("evtResultInitiateScan",function(scanNewDeviceRes) {
    //handle the scan response here
});

//call initiateScan
shdap.connection.initiateScan(2, 60);

//call initiateScanNewDevice
shdap.connection.initiateScanNewDevice(2, 60);
```

*Note:* For more details, refer the API Specification document.
6.2 Multiple Application Feature

Intel SHDAP components (open and horizontal) and APIs enable multiple applications to realize smart home goals quickly and easily. Intel SHDAP can execute multiple applications. The following are essential elements of the multiple-application feature.

- Each application contains an appId. The application is registered with Intel SHDAP by passing the appId via the appRegister() API.

- The Intel SHDAP SDK response is delivered only to the application that has made the request to the API.
  
  For example, App1 makes the call to InitiateScan() and gets the event response evtResultInitiateScan. The event is not delivered to App2.

- The smart home sensor/actuator/device events and alerts are delivered as notifications to all active applications.
  
  For example, the motion sensor alert is delivered to all applications.

- Rules are visible to all applications and can be modified by any application.

- All applications are allowed to call Scan/Add/Remove/Read/Write/Get Device Info APIs.

- All applications are allowed to change device alias names.

- Intel SHDAP does not allow simultaneous scans by two or more applications.

6.3 Group Management

Intel SHDAP SDK provides a set of APIs to group the devices. Group Management APIs enable you to logically group a set of devices that are scanned and paired in the home network, which can be used for Visual UI representations.

- Group Management APIs enable you to create, rename, and delete groups; and list created groups.

- You can add/remove the device to/from the group using the device id or alias.

- You can list devices in the group.
7.0 Intel® SHDAP SDK Sample Applications

Intel SHDAP SDK includes sample applications that provide reference implementations using the SDK APIs to help developers build innovative smart home applications quickly.

The SDK provides four types of sample applications:

1. HTML5-based sample user application
2. Node.js*-based sample applications
3. Sample applications that can be used through Intel® XDK
4. Cloud Connector sample applications

This section provides brief information for each sample application.

7.1 HTML5 Sample Application

The HTML5-based reference application uses APIs provided by Intel SHDAP SDK.

This application is written with the HTML5 markup language and Bootstrap* ([http://getbootstrap.com](http://getbootstrap.com)). The application communicates with the web server on the Gateway using the REST Interface, developed in Node.js environment.

Launching the application:

1. The HTML5 application is available in the following directory:
   `/home/shdap/Intel/shdap_sdk/sample_apps/mobile_apps/SmartHomeApp`
2. Find the IP address of the Gateway using the “ifconfig” command.
3. Launch a browser from either the mobile, tablet, or PC platform. It should be within the same network. Provide the Gateway IP address (with the “http://” prefix) at the address bar. The HTML application will be rendered in the browser.
4. Launch the application:
   `$ node SampleHomeApp.js`

7.1.1 Application Features

You can explore the following features using the reference application:

1. You can control the devices connected to the Gateway by clicking **I'M HOME**. This feature enables you to control the devices of different groups ("Room" is mentioned as a group in the application). Apart from basic turn-on and turn-off operations, the application enables you to perform device-supported operations.
2. You can perform different operations on the device by clicking **DEVICES**.
– **Scan and add device** enables you to scan different types of devices, for example, ZigBee*- and Z-Wave*-based devices, and add them to the Gateway. The application also enables you to assign aliases to the devices.
– **Add device to Room** enables you to add a device to a group, for example, Bedroom, Livingroom, and Kitchen.
– **Remove device from Room** enables you to remove a device from a group. This feature removes the device from the group, but the device is still connected to the Gateway so it can be added to a new group after being removed from the previous group.
– **Remove device from Gateway** enables you to remove a device from the Gateway.

3. You can perform group operations by clicking **GROUPS**:
– **Add Room** enables you to add a new group in the Gateway. After the group is created, you can add devices to it.
– **Remove Room** enables you to remove a group from the Gateway. After the group is deleted, the device assigned to the group can be used for different groups.
– **Rename Room** enables you to rename the group.

4. You can perform various operations for rules by clicking **RULES**:
– List all the rules
– Enable a rule
– Disable a rule
– Delete a rule
– Delete all rules

**Figure 2. Intel® SHDAP**

![Intel® SHDAP](image_url)
7.1.2 Making the Application as a Service

Create a service file to run SampleHomeApp as a service using the following steps. The service file called SampleHomeApp.service is created in this example. The following outlines the content of a typical service file for this configuration setup.

```
[Unit]
Description=Sample homeapp
After=network.target

[Service]
Type=simple
Environment="NODE_PATH=/usr/lib/node_modules"
ExecStart=/usr/bin/node
/home/shdap/Intel/shdap_sdk_pr0.82/shdap/sample_apps/mobile_apps
/SmartHomeApp/SampleHomeApp.js
Restart=always

[Install]
WantedBy=multi-user.target
```

1. Place the attached file `SampleHomeApp.service` in the folder `/lib/systemd/system/`

2. On the command prompt, execute the following commands:
   ```
   $ systemctl enable SampleHomeApp.service
   $ systemctl start SampleHomeApp.service
   ```

3. Reboot the board and execute the “systemctl” command as follows to check whether “SampleHomeApp” is running. If the service is up and running, the status command provides output similar to the following:
   ```
   $ systemctl status SampleHomeApp.service
   ```

   ![Output](https://via.placeholder.com/150)

4. To check the logs, execute the following command:
   ```
   $ journalctl -u SampleHomeApp -f
   ```
7.1.3 Enhancing the Application
The file SmartHomeApp/shdap/dist/js/createGroup.js contains a list of device types currently supported in the sample applications. More device types can be added as required.

7.2 Node.js*-Based Sample Applications
Intel SHDAP SDK sample applications developed using the Node.js environment serve as reference applications for developers to create innovative smart home applications and use cases.

This release provides two Node.js-based sample applications.

To run a sample application, log in as an “app” user from a new terminal.
$ ssh app@<IP address>

If you are connected through serial terminal and trying to run application.js, make sure the board is assigned with IP address (LAN / WiFi). This configuration is required for adding ZigBee devices.

To launch the connectivity sample application:
$ node application.js

To launch the rule engine sample application:
$ node applicationRuleEngine.js

7.2.1 Connectivity Sample Application
“application.js” showcases the usage of the following connectivity and device access APIs:

- initiateScan
- addDevice
- addDeviceByAlias
- readDeviceAttribute
- readDeviceAttributeByAlias
- writeDeviceAttribute
- writeDeviceAttributeByAlias
- removeDevice
- removeDeviceByAlias
- setDeviceAlias
- updateDeviceAlias
- findDevices

Log in as an "app" user on a new terminal to execute the connectivity sample application "application.js" with application user privileges.

```
$ ssh app@<IP address>
$ node application.js
```

### 7.2.2 Rule Engine Sample Application

`applicationRuleEngine.js` showcases the usage of the following Rule Engine APIs:

- addRule
- deleteRule
- enableRule
- disableRule
- modifyRule
- deleteAllRules
- listAllRules

Once the board boots up, the Intel SHDAP services are automatically started. Make sure that all services are running.

Log in as an "app" user on a new terminal to execute the rule engine sample application "applicationRuleEngine.js" with application user privileges.

```
$ ssh app@<IP address>
$ node applicationRuleEngine.js
```

Sample rules are provided with the rule engine sample application for reference. These rule files are installed to the following directory:

```
/home/shdap/Intel/shdap_sdk/sample_apps/sdk_apps.
```

Log in as an "shdap" user to access them. The following is a snippet of the "GEBulb.rule" rule.

```
ruleset: smartHomeDap_basic_rules_GEBulb
appid: sample_app
rule: simple_GEBulb_rule
{
  when
```
time_trigger: timer repeat "06:28:00"
if(1 == 1 )
then
    @change_state("oic:?ce524000011d4f8", {"value": 1})
else
end

To check the correctness of the rule grammar, validate the rule by copying it into the online PEG.js site at http://peg.js.org/

After running the registration of applicationRuleEngine.js, press “1” on the keyboard to select a rule from the addRule menu option to add to the rule database. In the preceding rule, oic:7ce524000011d4f8 refers to the device ID address. Replace this with the valid device ID (MAC address prefixed with “oic:”) before adding the rule to the database.

After adding the rule to the database, perform a read and write operation on the mentioned device so that the device facts get updated in the facts database. This will help rule execution.

For more information about the API, see Intel® Smart Home Development Acceleration Platform (Intel® SHDAP) Software Development Kit (SDK) SDK API Specification.

7.3 Intel® XDK Sample Applications

The following non-interactive applications can be executed using the Intel® XDK tool.

- applicationHelloWorld.js
- applicationDeviceOperation.js
- applicationGroupManagement.js
- applicationRuleEngine.js
- applicationRuleEngineSimpleComplex.js
- complexRuledoorLock.rule
- simpleDoorLock.rule

7.4 Cloud Connector Sample Applications

Note: The screenshots in this section are taken from Amazon Web Services* (AWS*) and Microsoft* Azure* cloud services portals at the time of document creation. Actual layout on these portals may be different.
Amazon Web Services* (AWS*) Sample Application

The Amazon Web Services* (AWS*) sample application demonstrates a way to register the Intel SHDAP Gateway device with the AWS cloud and to aggregate the sensor data for the sensors connected to the Gateway, onto the cloud.

At a high level, the following steps are required:

1. Sign up on the AWS portal and obtain a key pair to access the cloud programmatically.
2. Generate a certificate/key pair that is used by the AWS APIs to communicate sensor data to the cloud (a sample application is included in the package to generate this certificate pair).
3. The sample application using AWS APIs and certificates generated in step 2 sends the sensor data to the cloud.

The following sections explain each step in detail.

7.4.1.1 Sign Up and Acquire the Key Pair

1. Go to https://aws.amazon.com/ and follow the instructions to sign up.
2. Get your Access Key ID and Secret Access Key (key pair):
   
   a. Open the IAM console on AWS: Services > Security & Identity > IAM.
   b. In the navigation pane, choose Add User.
   c. Enter the User name.
   d. Select the access type as Programmatic access.
   e. Select Attach existing policy directly and search for AWSIoTFullAccess. Click NEXT to attach the policy to IAM.
   f. Upon completion of the user creation process, click the “Download csv” button to download the key pair.
7.4.1.2 Generate a Certificate/Key Pair

1. On the board, go to:

   
   /home/shdap/Intel/shdap_sdk/sample_apps/gateway_apps/CreateAwsKeys

2. Open the index.js file and change the value of "thingName" in the file to the desired name to appear on the cloud, for the Gateway. For example:

   thingName: 'GW_03'

3. From the .csv file downloaded in Section 7.4.1.1, populate the values of parameters ACCESS_KEY_ID and SECRET_ACCESS_KEY in the index.js file.

4. Close the index.js file and run it:

   node index.js

Output similar to the following appears:

```javascript
{
  thingName: 'GW_03',
  thingArn: 'arn:aws:iot:us-west-2:416569913032:thing/GW02'
}
```

```javascript
{
  certificateArn: 'arn:aws:iot:us-west-2:416569913032:cert/c7e43440ee8ae8e5834c19239704af77dc7c7d1933df92d188bd64d4d701efd5',
  certificateId: 'c7e43440ee8ae8e5834c19239704af77dc7c7d1933df92d188bd64d4d701efd5',
  certificatePem: '-----BEGIN CERTIFICATE-----

   nMIIDWjCCAkKgAwIBAgIVAIjbbbPhnFUASKyQs8vgrKIb80imMA0GCSqGSIb3</string>
   DQEB\n
   b3DQEB\n
   nWIAMEExz5BJBgNVBAsMQkFtYXpvbiBXZWIgU2Ymd1jZXMGtz1BbWF6b24uY29t

   nIElUyY4gTD1TWFDodGxlIFNUPVdhc2hbmd0b24gQz1VUz

   AefW6x0NjExMjAxNDEwInMzNaFw00OTEyMzEyMzU5NaFw00OTEyMzEyMzU5

   nM51aMB4uHDAAAgNVBAMM

   E0FXUYB3b1QgQ2VydGlmaWNh\n
   ndGUwggEiMA0GCSqGSIb3DQEBAQUAA4IBAgAwggEkAoIABBCKxsIOQOa+LJncBh

   nE2TUTxqzAvosspGR6At5oITVc8TmJ1B

   6opm15Teya4HkNN2zDUbU5FHPGpaDhCEbW1ZJ2Qs

   nHjc6NhivVBPj1KSJ

   17gcQOFnbbXrvmYgaBYlzYMPFJCCm13PqdyjmBZnT6/iygx\n
   nEW1ldJb5MsG5HP48RgsPbSxjht/0W5qIP+gMuhjCBzV5BVxOyn9tM5+nYkz

   nS5Y1uWHRx2w/88NPQ1NH1KrO/BXhY7FHXjKqfNMMeOOG3bqakE4cDn2wYg3AtUXw

   nEFOFqMBAAAgjYDBeMB9GA1UDiWQYMBaAFAIANK91/3cxQgQWGX4iwTWod

   NHMBOG\n
   nA1UDqQWB8QYnYc+d+kPGR/nUXs7uyE7n/CyiczAMBagNHRNBaFeA

   AjAAM4G4A1Ud/nDwEB/wQEAwIHgDBNbgkhiG9w0BAQsFAAOCAQEOAS2XoFK

   5S17dAxtj8I1NAN7/nA53wCoqQNzkqFQFcsSr5bp4vkOJL0L0UnNgEqwyhLt

   TW9mdwVWQUPFXXb5E74+eFA\n
   ng/vrG9eXE1ksmIN2W4c4krbK0XTH2dU

   NeI9jFvwhhd10/E/Q9J7Z6UsjUg56kr/n3181C1790Rc/OHAg8eELZ4WbppZ

   xZAAx90Hyg5y1JLZ121e61jFwqH40bij+Fh0rnf+t3N3JXQA/7SeraCA/J

   MeO+kO0NRu3ioLdKtYwt7j3xn0jik5S12Z1rw6mGldb/n29U9q6GDR93qQj
```
QkP5zrPlHyRk3zotKFciJVF1LN4xRA1kQkhqgp3Trv1g==

keyPair: {'Public Key': '
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAisbCDjkKAPiyZ3AQ4RNK/n1e8aswL6LRK1EeQN6d9RLGJ/Cay7rvni4hWqt7+6FL86XAnBN2wZ09ixa1Qouf/n1YW2z3mPVmEW2fKXEB2yOqQz10UR3mu8BDTmXcAbhU6Rhj6Wg4XBGlyWYULB43/n0jYyr1qaY9SKide6EIDRZ1I167zGIggWJc2DdxSQgptdz6ncp5gWZ0+4vsoMRpFpNYN+TcLJuT+PIEabD2ksbY4bf9M/nEqyd/oDLOyWgc1QVcTspwPBtNpfp2Cs+ci/nLh0cdeP/PDT0JTR5qqzwWV4OxYrcSg0n7Hjhtj26mpBJeHI5+GcGINwLVF8BBT/nhQIDAQAB-----END PUBLIC KEY-----
', 'PrivateKey': '
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAisbCDjkKAPiyZ3AQ4RNK1E8aswL6LRK1EeQN6d9RLGJ/Cay7rvni4hWqt7+6FL86XAnBN2wZ09ixa1Qouf/n1Y2z3mPVmEW2fKXEB2yOqQz10UR3mu8BDTmXcAbhU6Rhj6Wg4XBGlyWYULB43/n0jYyr1qaY9SKide6EIDRZ1I167zGIggWJc2DdxSQgptdz6ncp5gWZ0+4vsoMRpFpNYN+TcLJuT+PIEabD2ksbY4bf9M/nEqyd/oDLOyWgc1QVcTspwPBtNpfp2Cs+ci/nLh0cdeP/PDT0JTR5qqzwWV4OxYrcSg0n7Hjhtj26mpBJeHI5+GcGINwLVF8BBT/nhQIDAQAB-----END RSA PRIVATE KEY-----

arn:aws:iot:us-west-2:241656991302:cert/c7e43440ee8ae8e5834c19239704af77dc7c7d1933df92d188bd64d4701efd5
'}
5. Copy the contents of the section "certificatePem:" from the preceding output into the “/home/shdap/Intel/shdap_sdk/sample_apps/gateway_apps/AwsSample/certs/certificate.pem.crt” file. Replace all instances of “\n” with actual line breaks in the certificate.pem.crt file.

6. Copy the contents of the section “PrivateKey:" from the preceding output into the “/home/shdap/Intel/shdap_sdk/sample_apps/gateway_apps/AwsSample/certs/private.pem.key” file. Replace all instances of “\n” with actual line breaks in the private.pem.key file.

7.4.1.3 Run the Sample Application

1. Go to

```
/home/shdap/Intel/shdap_sdk/sample_apps/gateway_apps/AwsSample
```

2. Put a ZEN Thermostat* using ZigBee solution, in pairing mode

3. Run the sample application

```
node index.js
```

The application scans and adds the thermostat to the Intel SHDAP Gateway and begins to send temperature notifications to the cloud. You can see the notifications by checking the publish message on the console.

4. This sample works with the ZEN Thermostat only. Modify the `index.js` file as follows for other sensor devices:

   On line 26, change the scan type according to the desired device type to scan:

   ```javascript
   scan.connection.initiateScan(2, 123);
   ```

   1-Bluetooth* low energy (LE) device, 2-ZigBee device, 3-Z-Wave device, 4-Philips* Hue device : 123-timeout

   On line 37, change the property name from “heatingSetPoint” to the property name of the desired device to test:

   ```javascript
   scan.deviceaccess.readDeviceAttribute(deviceIds,"heatingSetPoint");
   ```

   Do the same on line 47:

   ```javascript
   var propTemp=properties.heatingSetPoint
   ```

   On line 51, change the JSON format to suit the desired device’s property:

   ```javascript
   var data = JSON.stringify({ 'deviceId': ConnectedDeviceId,'msgTyp':"temperature_msg","temp":propTemp });
   ```
To check whether the device is registered on the AWS site, log in to the AWS IOT Console (choose Services > Internet of Things > AWS IoT).

A box with the name given in the certificate generator application is shown, in this case, GW_03.

7.4.2 Microsoft* Azure* Sample Application

This sample application demonstrates a way to register the Intel SHDAP Gateway device with the Microsoft Azure cloud and to aggregate the sensor data for the sensors connected to the Gateway, onto the cloud.

To begin using this application, sign up on the Microsoft Azure portal (https://azure.microsoft.com) and generate a login ID.

7.4.2.1 Create an IoT Hub

Create an IoT hub to connect devices. The following steps show how to complete this task through the Microsoft Azure portal.

1. Sign in to the Microsoft Azure Portal.
2. In the Jumpbar, click New > Internet of Things > Azure IoT Hub.
3. In the IoT hub blade, choose the configuration for the desired IoT hub.

   - In the Name box, enter a name for the desired IoT hub. If the Name is valid and available, a green check mark appears in the Name box.
   - Select a pricing and a scale tier. This tutorial does not require a specific tier. For this sample application, use the free F1 tier.
In Resource group, create a new resource group, or select an existing one. For more information, see the Microsoft Azure site to use resource groups to manage Microsoft Azure resources.

In Location, select the location to host the desired IoT hub. For this sample application, choose the nearest location.

After choosing the IoT hub configuration options, click Create. It can take a few minutes for Microsoft Azure to create the IoT hub. To check the status, monitor the progress on the Startboard or in the Notifications panel.

4. After the IoT hub has been created successfully, click the new tile for the IoT hub in the Microsoft Azure portal to open the blade for the new IoT hub. Note the Hostname, and then click Shared access policies.
5. In the Shared access policies blade, click the iotubowner policy, and then copy and note the connection string in the iotubowner blade.

![Image of Shared access policies blade]

7.4.2.2 Run a Sample Application

When the setup on Microsoft Azure portal is done, a few parameters need to be set in the Microsoft Azure sample application in the Gateway.

1. The sample application can be found at:
   `/home/shdap/Intel/shdap_sdk/sample_apps/gateway_apps/AzureSample`

2. Make the following changes in the `index.js` file in this location:
   - At line 34, assign `IotHubConnectionString = <connection string marked in the preceding screenshot>`
   - At line 38, assign `device.deviceid = <desired name>`

3. Put the ZEN Thermostat in pairing mode and run the application:
   ```javascript
   node index.js
   ```

   The application scans and adds the thermostat to the Intel SHDAP Gateway and begins to send temperature notifications to the cloud. You can see the notifications by checking the publish message on the console.

   Similar to the AWS sample application, this application is also configured by default to send temperature notifications from the ZEN Thermostat to the Microsoft Azure cloud. This application can be modified for other devices by following the steps in Section 7.4.1.3.

   The following steps show how to see the device on the Microsoft Azure portal:
   1. Go to the created Microsoft Azure IoT Hub.
   2. Check the donut chart in the Usage block to see how many devices are connected to the IoT Hub.
3. Click the **Devices** tab (highlighted in yellow in the following screenshot) to check whether the device is created.

The created device appears on the right panel.

![Device Creation Screenshot](image)

### 7.5 Adding a New Smart Home Application

Using the Node.js-based Intel SHDAP APIs, you can develop innovative smart home applications. This section describes a few configurations required for adding a new application to an Intel SHDAP Gateway.

Each application developed must be associated with a unique app-id and must register with Intel SHDAP via the appRegister() API call before using other APIs.

The following are a few points about the app-id:

- **Configure the** `config.txt` **file at the following location before using the app-id:**
  ```
  /home/shdap/Intel/shdap_sdk/bin/config.txt
  ```

- The following app-ids are enabled by default and assigned to the sample applications:
  - 1
  - 2
  - xdk_ado
  - xdk_agm
  - xdk_ahw
  - xdk_are
  - xdk_aresc
  - tsa_a
  - tsa_are
  - mobileapp_ri
  - sampleapp_ri
• The following is a sample entry of an app-id in the config.txt file:

  ```json
  {"org-id":"8", "id":"tsa_a", "description": "Intel sdk api test app", "topics": ["re_req_addRule", "re_req_deleteRule"]}
  ```

• About 50 app-ids can be configured in the config.txt file.
8.0 Configuring Philips* Hue Bridge Plugin

The following are steps to add the Philips* Hue bridge into Intel® SHDAP SDK.

8.1 Configuring Philips* Hue Bridge

1. Get the Philips Hue bridge IP address.
2. Using the IP address, type the following URL in a browser:

   The following page opens.

   ![CLIP API Debugger](image)

3. In the CLIP API Debugger URL, type /api.
4. Press the Link button on the Philips Hue bridge (top button on the bridge).
5. In the Message Body window, paste the following line:
   
   ```json
   {"devicetype":"my_hue_app#iphone peter"}
   ```
6. Press the POST button.
Response:
[
{
  "success": {
    "username": "1028d66426293e821ecfd9ef1a0731df"
  }
}
]

7. Make sure that the generated user name value is in a single line shown and highlighted as follows:
   o "username":
      "1028d66426293e821ecfd9ef1a0731df"

See the following for details:
http://www.developers.meethue.com/documentation/getting-started
8.2 Updating Bridge Details

Update the generated user name and MAC ID in the `hue_auth_json.txt` file located at `/home/shdap/Intel/shdap_sdk/bin`. User name is the generated user name value from Section 8.1. MAC ID is a unique ID from the Philips Hue bridge’s label print.

The following example shows the updated user name and ID in the `hue_auth_json.txt` file.

```json
[
  {
    "id": "001788fffe1CAF84",
    "username": "1028d66426293e821ecfd9ef1a0731df"
  }
]
```

**Note:** Do not replace the complete MAC ID, just replace the last 6 hex digits. Having the four hex digits "fffe" (string format) in the middle of the ID is correct. Do not delete it from the original file ID string format.

**Note:** The label print of the Philips Hue bridge contains either 12 or 6 hex digits, depending on the production phase. The following image shows a model with 12 hex digits printed on the label.

![Image of Philips Hue Bridge Label](image_url)

**Note:** After saving the `hue_auth_json.txt` file, reboot the board.
9.0  **Supported Device Configurations**

9.1  **ZigBee* Device Configuration**

The following is a list of supported clusters.

**Table 4.  List of Supported Clusters**

<table>
<thead>
<tr>
<th>S.N</th>
<th>Supported Clusters</th>
<th>ClusterId</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Basic</td>
<td>0x0000</td>
</tr>
<tr>
<td>2</td>
<td>Identify</td>
<td>0x0003</td>
</tr>
<tr>
<td>3</td>
<td>On/Off</td>
<td>0x0006</td>
</tr>
<tr>
<td>4</td>
<td>Level Control</td>
<td>0x0008</td>
</tr>
<tr>
<td>5</td>
<td>Door Lock</td>
<td>0x0101</td>
</tr>
<tr>
<td>6</td>
<td>Thermostat</td>
<td>0x0201</td>
</tr>
<tr>
<td>7</td>
<td>Fan Control</td>
<td>0x0202</td>
</tr>
<tr>
<td>8</td>
<td>Color Control</td>
<td>0x0300</td>
</tr>
<tr>
<td>9</td>
<td>Temperature Measurement</td>
<td>0x0402</td>
</tr>
<tr>
<td>10</td>
<td>Relative Humidity Measurement</td>
<td>0x0405</td>
</tr>
<tr>
<td>11</td>
<td>IAS Zone</td>
<td>0x0500</td>
</tr>
<tr>
<td>12</td>
<td>Electrical Measurement</td>
<td>0x0b04</td>
</tr>
<tr>
<td>13</td>
<td>Power Configuration¹</td>
<td>0x0001</td>
</tr>
<tr>
<td>14</td>
<td>IAS Warning Device</td>
<td>0x0502</td>
</tr>
</tbody>
</table>

**NOTE:**
1. In the Power Configuration cluster, only the “BatteryPercentageRemaining” attribute is supported.

The following are properties supported in implemented clusters:

**Table 5.  Properties Supported in Implemented Clusters**

<table>
<thead>
<tr>
<th>#</th>
<th>Cluster</th>
<th>Supported Property</th>
<th>Tested Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Basic</td>
<td>ZCLVersion, powerSource, secondarySourcePresent</td>
<td>All</td>
</tr>
<tr>
<td>2</td>
<td>Identify</td>
<td>identifying, identifyTimeout</td>
<td>All</td>
</tr>
<tr>
<td>3</td>
<td>On/Off</td>
<td>value</td>
<td>Philips* Hue Bulb, OSRAM* Bulb</td>
</tr>
</tbody>
</table>
## Supported Device Configurations

<table>
<thead>
<tr>
<th>#</th>
<th>Cluster</th>
<th>Supported Property</th>
<th>Tested Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Level Control</td>
<td>currentLevel, moveToLevel, levelTransitionTime, moveMode, moveRate, moveToLevelWithOnOff, moveModeWithOnOff, stop, stepMode, stepSize, stepTransitionTime, stepModeWithOnOff</td>
<td>GE* Bulb, CREE* Bulb, Centralite* Smart Outlet, Quirky-GE* Smart Switch, GE Smart Outlet, Philips Hue Bulb, OSRAM Bulb, GE Bulb, CREE Bulb</td>
</tr>
<tr>
<td>5</td>
<td>Door Lock</td>
<td>lockState</td>
<td>Yale* Door Lock.</td>
</tr>
<tr>
<td>6</td>
<td>Thermostat</td>
<td>LocalTemperature, coolingSetPoint, heatingSetPoint</td>
<td>ZEN Thermostat*, Centralite* Pearl Thermostat</td>
</tr>
<tr>
<td>7</td>
<td>Fan Control</td>
<td>speed, range</td>
<td>ZEN Thermostat, Centralite Pearl Thermostat</td>
</tr>
<tr>
<td>8</td>
<td>Color Control</td>
<td>hue, saturation</td>
<td>Philips Hue Bulbs</td>
</tr>
<tr>
<td>9</td>
<td>Temperature Measurement</td>
<td>measuredTemperatureValue, minMeasuredTempValue, maxMeasuredTempValue</td>
<td>Centralite Motion Sensor, Centralite Water Leak, Centralite Door Sensor, Centralite Temperature and Humidity Sensor, SmartEnit* Motion Sensor, Nyce* Ceiling Motion Sensor</td>
</tr>
<tr>
<td>10</td>
<td>Relative Humidity Measurement</td>
<td>Humidity, minHumidity, maxHumidity</td>
<td>Nyce Ceiling Motion Sensor</td>
</tr>
<tr>
<td>11</td>
<td>IAS Zone</td>
<td>value1, value2, motion1, motion2, doorsense1, doorsense2</td>
<td>Centralite Motion Sensor, Centralite Water Leak, Centralite Door Sensor, SmartEnit Motion Sensor, Nyce Ceiling Motion Sensor, Philio Smoke Sensor, Philio Water Leak Sensor</td>
</tr>
</tbody>
</table>
## Supported Device Configurations

<table>
<thead>
<tr>
<th>#</th>
<th>Cluster</th>
<th>Supported Property</th>
<th>Tested Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tampered</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>lowBattery</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>deviceFailure</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>mainsFault</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Electricity</td>
<td>measurementType</td>
<td>Centralite Smart Outlet</td>
</tr>
<tr>
<td></td>
<td>Measurement</td>
<td>acFrequency</td>
<td>Philio Smart Plug</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rmsVoltage</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>rmsCurrent</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>activePower</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>acCurrentMultiplier</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>acCurrentDivisor</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>acPowerMultiplier</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>acPowerDivisor</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Power</td>
<td>BatteryLevel</td>
<td>Yale Door Lock</td>
</tr>
<tr>
<td></td>
<td>Configuration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>IAS Warning</td>
<td>maxDuration</td>
<td>Philio Multi Sound Siren</td>
</tr>
<tr>
<td></td>
<td>Device</td>
<td>warningInMode</td>
<td></td>
</tr>
</tbody>
</table>

### 9.1.1 Using MMB Sample Application

The MMB sample application demonstrates the ZigBee* coordinator interface.

The following commands can be issued to use the sample application:

**Table 6. Commands for the Sample Application**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>Runs module info request</td>
</tr>
<tr>
<td>N</td>
<td>Runs network status test</td>
</tr>
<tr>
<td>P</td>
<td>Opens permit window for 60 seconds</td>
</tr>
<tr>
<td>D</td>
<td>Prints device table</td>
</tr>
<tr>
<td>Z</td>
<td>Runs discovery on nodes</td>
</tr>
</tbody>
</table>

The following steps check the supported clusters:

1. Run the MMB test application:
   ```
   $ cd /home/shdap/Intel/shdap_sdk/tools
   $ ./mmbSample /dev/ttyS0
   ```
2. Run commands "M", "N", and then "P".
3. Turn the end device into the discoverable mode.
4. Once the device is discovered, run "Z" to get endpoint and cluster details from the device.
5. Run "D" to print the device table that has the device endpoint and cluster details.

### 9.1.1.1 Checking Whether a New Device Is Supported by the ZigBee* Plugin

1. Find all clusters that the new device supports (one way is to use the MMB sample application, or refer to the device specifications).
2. If the device has at least one cluster implemented in the ZigBee* Plugin, the device is supported by Intel SHDAP. The device can be controlled only on the properties that come under the supported clusters.

### 9.2 Z-Wave* Device Configuration

The following is a list of supported command classes:

**Table 7. List of Supported Command Classes**

<table>
<thead>
<tr>
<th>S.N</th>
<th>Supported Command Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>COMMAND_CLASS_BASIC</td>
</tr>
<tr>
<td>2.</td>
<td>COMMAND_CLASS_SENSOR_BINARY</td>
</tr>
<tr>
<td>3.</td>
<td>COMMAND_CLASS_BATTERY</td>
</tr>
<tr>
<td>4.</td>
<td>COMMAND_CLASS_SENSOR_MULTILEVEL</td>
</tr>
<tr>
<td>5.</td>
<td>COMMAND_CLASS_SWITCH_BINARY</td>
</tr>
<tr>
<td>6.</td>
<td>COMMAND_CLASS_DOOR_LOCK</td>
</tr>
<tr>
<td>7.</td>
<td>COMMAND_CLASS_ALARM</td>
</tr>
<tr>
<td>8.</td>
<td>COMMAND_CLASS_POWERLEVEL</td>
</tr>
<tr>
<td>9.</td>
<td>COMMAND_CLASS_METER</td>
</tr>
<tr>
<td>10.</td>
<td>COMMAND_CLASS_PROTECTION</td>
</tr>
<tr>
<td>11.</td>
<td>COMMAND_CLASS_SWITCH_MULTILEVEL</td>
</tr>
<tr>
<td>12.</td>
<td>COMMAND_CLASS_WAKE_UP</td>
</tr>
<tr>
<td>13.</td>
<td>COMMAND_CLASS_THERMOSTAT_SETPOIN</td>
</tr>
<tr>
<td>14.</td>
<td>COMMAND_CLASS_THERMOSTAT_FAN_MODE</td>
</tr>
<tr>
<td>15.</td>
<td>COMMAND_CLASS_THERMOSTAT_FAN_STATE</td>
</tr>
<tr>
<td>16.</td>
<td>COMMAND_CLASS_THERMOSTAT_MODE</td>
</tr>
<tr>
<td>17.</td>
<td>COMMAND_CLASS_THERMOSTAT_OPERATING_STATE</td>
</tr>
<tr>
<td>18.</td>
<td>COMMAND_CLASS_INDICATOR</td>
</tr>
<tr>
<td>19.</td>
<td>COMMAND_CLASS_CLOCK</td>
</tr>
<tr>
<td>20.</td>
<td>COMMAND_CLASS_FIRMWARE_UPDATE_MD</td>
</tr>
</tbody>
</table>
The following is a list of command class names and supported properties:

Table 8. Command Class Names and Supported Properties

<table>
<thead>
<tr>
<th>#</th>
<th>Command Class</th>
<th>Supported Property</th>
<th>Tested Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>COMMAND_CLASS_BASIC</td>
<td>Value</td>
<td>Aeotec* Water Sensor</td>
</tr>
<tr>
<td>2</td>
<td>COMMAND_CLASS_SENSOR_BINARY</td>
<td>value_notify</td>
<td>Aeon* D/W Sensor, Schlage* D/W Sensor, Ecolink* D/W Sensor, Everspring* D/W Sensor, Ecolink Motion Sensor, Aeotec Multi-Sensor</td>
</tr>
<tr>
<td>4</td>
<td>COMMAND_CLASS_SENSOR_MULTILEVEL</td>
<td>temperature,</td>
<td>Aeotec Multi-Sensor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>humidity, luminance,</td>
<td>FortrezZ Water Sensor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>power, generalpurpose,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>winddirrection,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>velocity, atmosphere,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>barometer</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>COMMAND_CLASS_SWITCH_BINARY</td>
<td>switchstate</td>
<td>GE* Outlet, Leviton* Outlet, TKB* Smart Switch, Aeon Smart Switch</td>
</tr>
<tr>
<td>6</td>
<td>COMMAND_CLASS_DOOR_LOCK</td>
<td>lockState</td>
<td>Yale Door Lock, SCHLAGE Door Lock</td>
</tr>
<tr>
<td>7</td>
<td>COMMAND_CLASS_ALARM</td>
<td>lowBattery, tempered,</td>
<td>Yale Door Lock, Aeon D/W Sensor, SCHLAGE D/W Sensor, SCHLAGE Door Lock, Vision* D/W Sensor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SmokeAlert, waterLeak,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>intrusion</td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>Command Class</td>
<td>Supported Property</td>
<td>Tested Devices</td>
</tr>
<tr>
<td>---</td>
<td>--------------------------------</td>
<td>---------------------------------------------------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>glassBreak, heatState, gasDetected, alarmClock, tempRise, acConnectionStatus, powerState, systemcFail, emergency, userCodeState</td>
<td>Vision Vibration Sensor First Alert Smoke Detector FortrezZ Water Sensor</td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>COMMAND_CLASS_POWER LEVEL</td>
<td>Power</td>
<td>GE Outlet, Leviton Outlet, GE Dimmer Switch, Aeon Smart Switch, TKB Switch, TKB Dimmer Switch, Cooper* Dimmer Switch</td>
</tr>
<tr>
<td>9.</td>
<td>COMMAND_CLASS_METER</td>
<td>electricPowerKW H, electricPowerKVA H, electricPowerW, electricPowerPulse, electricPowerVolt, electricPowerAmp, electricPowerFactor</td>
<td>Aeon Micro Smart Switch NorthQ Power Reader, Aeon Energy Meter, Aeon Micro Smart Dimmer Switch</td>
</tr>
<tr>
<td>10.</td>
<td>COMMAND_CLASS_PROTECTION</td>
<td>locProtectionStatus</td>
<td>Living Connect Radiator Thermostat</td>
</tr>
<tr>
<td>11.</td>
<td>COMMAND_CLASS_SWITCH_MULTILEVEL</td>
<td>dimmingsetting</td>
<td>GE Dimmer Switch, Cooper Dimmer Switch, Leviton Dimmer Switch, TKB Dual Dimmer Switch, Aeon Smart Dimmer Switch</td>
</tr>
<tr>
<td>12.</td>
<td>COMMAND_CLASS_WAKE_UP</td>
<td>Interval</td>
<td>NorthQ Power Reader, Aeon D/W Sensor, Aeon Multi Sensor</td>
</tr>
<tr>
<td>13.</td>
<td>COMMAND_CLASS_INDICATOR</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>14.</td>
<td>COMMAND_CLASS_CLOCK</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
9.2.1 Adding Support for a New Vendor Z-Wave* to SDK

Any new Z-Wave* device that is not in the list of devices in this document can work with Intel SHDAP if the device supports a subset of the command classes listed in Table 8.

The following are configurations to enable a new device:

1. The Z-Wave plugin uses the configuration file `zwave_device_info.txt` to maintain vendor data such as vendor ID, vendor type, and product ID.

This configuration file is placed in `/home/shdap/Intel/shdap_sdk/bin` by the installer script with required group and permissions.

File format:

```json
[

{
  "vid" : 134,
  "vtype" : 2,
  "pid" : 4,
  "vname" : "AeonLabs",
  "devicetype" : "oic.d.sensor.door"
}
]```
Note: vid, vtype, pid, and vname denote vendor ID, vendor type, product ID, and vendor name, respectively.

Table 9. OIC Device Type Mapping

<table>
<thead>
<tr>
<th>S.N.</th>
<th>Device Name</th>
<th>OIC Device Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AEON* Door Window Sensor</td>
<td>&quot;oic.d.sensor.door&quot;</td>
</tr>
<tr>
<td>2</td>
<td>AEON Multi Sensor</td>
<td>&quot;oic.d.sensor.multi&quot;</td>
</tr>
<tr>
<td>3</td>
<td>Aeotec* Water Sensor</td>
<td>&quot;oic.d.sensor.water&quot;</td>
</tr>
<tr>
<td>4</td>
<td>GE* Smart Outlet</td>
<td>&quot;oic.d.switch&quot;</td>
</tr>
<tr>
<td>5</td>
<td>GE Dimmer Switch</td>
<td>&quot;oic.d.switch&quot;</td>
</tr>
<tr>
<td>6</td>
<td>Yale* Door Lock</td>
<td>&quot;oic.d.doorlock&quot;</td>
</tr>
<tr>
<td>7</td>
<td>Cooper* Dimmer Switch</td>
<td>&quot;oic.d.switch&quot;</td>
</tr>
<tr>
<td>8</td>
<td>Ecolink* Door Window Sensor</td>
<td>&quot;oic.d.sensor.door&quot;</td>
</tr>
<tr>
<td>9</td>
<td>Ecolink Motion Sensor</td>
<td>&quot;oic.d.sensor.motion&quot;</td>
</tr>
<tr>
<td>10</td>
<td>Levtion* Outlet</td>
<td>&quot;oic.d.switch&quot;</td>
</tr>
<tr>
<td>11</td>
<td>Everspring* D/W Sensor</td>
<td>&quot;oic.d.sensor.door&quot;</td>
</tr>
<tr>
<td>12</td>
<td>First Alert* Smoke Sensor</td>
<td>&quot;oic.d.sensor.smoke&quot;</td>
</tr>
<tr>
<td>13</td>
<td>FortrezZ* Water Sensor</td>
<td>&quot;oic.d.sensor.water&quot;</td>
</tr>
<tr>
<td>14</td>
<td>Schalge* Door lock</td>
<td>&quot;oic.d.doorlock&quot;</td>
</tr>
<tr>
<td>15</td>
<td>Living Connect* Thermostat</td>
<td>&quot;oic.d.thermostat&quot;</td>
</tr>
<tr>
<td>16</td>
<td>Aeon* Energy Meter</td>
<td>&quot;oic.d.energymeter&quot;</td>
</tr>
</tbody>
</table>

2. If device information is not present in the file, the plugin displays the following error at the time of scan:

   "Error: Data not found in device config list"

   To check the device vendor information, check for the following information in the plugin's log:

   "Info: Data received from End Device VID : VTYPE: PID:"

   Alternatively, check for information available at http://www.pepper1.net/zwavedb/

3. Before adding a new vendor device, stop the Intel SHDAP service.

4. Include the device information in the `zwave_device_info.txt` file, as mentioned in Section 9.2.1.
5. Restart the Intel SHDAP service.

9.2.2 Configuring Plugin to Map Vendor-Specific Data to Z-Ware Specified Information

Every device provides vendor-specific information, for example, alarm type and alarm level. The information must be mapped to the respective Z-Ware specified information, so that the plugin can identify the type of alert.

The Z-Ware library uses the `zwave_device_rec.txt` configuration file that is loaded during Z-Wave module initialization, to map the data. The configuration file is in JSON format and located at `/home/shdap/Intel/shdap_sdk/bin`. The file contains vendor information and device interface details.

Whenever there is a new device, fill the vendor-specific alarm type and alarm level (obtained from the device datasheet) in the `zwave_device_rec.txt` file for the library to map the data.

Configuration file sample:

```json
{
  "_comment": "First Alert Smoke & CO Alarm Detector",
  "vid":312,
  "ptype": 1,
  "pid":2,
  "category":1,
  "ep": [
    {
      "epid":0,
      "interface": [
        {
          "if_type": 1,
          "grp_id": [1]
        },
        {
          "if_type": 3,
          "bin_sensor_type": 2
        },
        {
          "if_type": 6,
          "alarm": [
            {
              "alarm_type": 12,
              "alarm_type_name": "Smoke alarm",
              "alarm_level_rec": [
                {
                  "alarm_level": 255,
                  "zw_alarm_event": 3,
                  "alarm_level_name": "Smoke alarm test"
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```
Table 10. Z-Wave Device Category

<table>
<thead>
<tr>
<th>S.N.</th>
<th>Device Category</th>
<th>Z-Wave Specified Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>General-purpose sensor</td>
<td>1</td>
</tr>
<tr>
<td>2.</td>
<td>Smoke sensor</td>
<td>2</td>
</tr>
<tr>
<td>3.</td>
<td>CO sensor</td>
<td>3</td>
</tr>
<tr>
<td>4.</td>
<td>CO2 sensor</td>
<td>4</td>
</tr>
<tr>
<td>5.</td>
<td>Heat sensor</td>
<td>5</td>
</tr>
<tr>
<td>6.</td>
<td>Water sensor</td>
<td>6</td>
</tr>
<tr>
<td>7.</td>
<td>Freeze sensor</td>
<td>7</td>
</tr>
<tr>
<td>8.</td>
<td>Tamper sensor</td>
<td>8</td>
</tr>
<tr>
<td>9.</td>
<td>Aux sensor</td>
<td>9</td>
</tr>
<tr>
<td>10.</td>
<td>Door/Window sensor</td>
<td>10</td>
</tr>
<tr>
<td>11.</td>
<td>Tilt sensor</td>
<td>11</td>
</tr>
<tr>
<td>12.</td>
<td>Motion sensor</td>
<td>12</td>
</tr>
<tr>
<td>13.</td>
<td>Glass break sensor</td>
<td>13</td>
</tr>
</tbody>
</table>

Follow these steps whenever there is a new device:

1. Get its vendor ID (vid), vendor type (vtype), and product ID (pid) from the datasheet and create a new JSON object.
2. Get the vendor-specific alarm type (alarm_type) and alarm level (alarm_level) from the datasheet and fill in the JSON object.
3. Check for the respective Z-Wave alarm type (zw_alarm_type) and alarm level (alarm_level_name) from the Z-Wave API reference document (ins13173-2_zwave_ce_c_api_-_user_guide), and fill in the JSON object.
4. Update the “category” in the JSON object by referring to Table 10.
5. Restart the Intel SHDAP services.

9.2.3 Configuring the Z-Wave Device Parameter

Every end device comes with specific configuration settings to enable some device features.
For example, the Aeon Multi-Sensor, as per default configuration settings, does not send automatic reports for temperature, luminance, humidity, and battery. Configure the sensor using parameter 101 to enable automatic reports for these properties.

- The configuration parameter and its value are specific to end devices. See the device's datasheet for details.
- The Z-Ware library maintains a device database file (zwave_device_rec.txt in Section 9.2.2) where a user can provide device configuration details (parameter number, size, and parameter value). Whenever the Z-Ware client is initiated and the user pairs that device with stack, it will be configured with new values.

Z-Ware Database File Format:

```json
{
  "device_records": [
    {
      "_comment": "Aeon labs Multi Sensor",
      "vid": 134,
      "ptype": 2,
      "pid": 5,
      "category": 11,
      "ep": [
        {
          "epid": 0,
          "interface": {
            "if_type": 1,
            "grp_id": [1]
          },
        },
        {
          "if_type": 2,
          "config": [
            {
              "param_num": 5,
              "param_size": 1,
              "param_val": 2
            },
            {
              "param_num": 101,
              "param_size": 4,
              "param_val": 225
            }
          ]
        },
        {
          "if_type": 3,
          "bin_sensor_type": 2
        }
      ]
    }
  ]
}
```
In the device database file, interface details (for example, bsensor, config, group, and alarm) can be provided. This file is used mainly to configure a device and map vendor-specific data to the Z-Ware specified format.

**Steps to Configure a Device:**

The following are steps to configure a Z-Wave end device:

1. Stop the Intel SHDAP services (TAL, broker, ppm, and root).
2. Open the `zwave_device_rec.txt` file in the following directory:
   
   ```
   /home/shdap/Intel/shdap_sdk/bin
   ```
3. Create a device database entry as mentioned in the preceding sample format.
4. For each device, mention the vendor ID (vid), vendor type (vtype), and product ID (pid), so that the library can identify a specific device.
5. Under the “interface” key, create one JSON entry for “if_type” 2 that is for “config”. This “config” key is the combination of the JSON pair and each JSON pair contains “param_num”, “param_size” and “param_val”.
6. Under "interface", create "if_type": 2. Under "if_type": 2, create "config". Under "config", create the JSON object. For the JSON object, insert the "param_num", "param_size", and "param_val" values (get these values from the device datasheet).

Start the Intel SHDAP service and add the device so that the library can configure it automatically with the provided configuration details in the `zwave_device_rec.txt` file.
10.0 Supported Devices

The following is a list of devices tested in this release:

Table 10. List of Devices Tested in This Release

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Manufacturer</th>
<th>Protocol</th>
<th>Properties</th>
<th>Read-only</th>
<th>Read-Write</th>
<th>Notify</th>
</tr>
</thead>
<tbody>
<tr>
<td>Door/Window Sensor</td>
<td>AEON* labs</td>
<td>Z-Wave*</td>
<td>value (0/1) batterystate (0-100)</td>
<td></td>
<td>value_notify (0/1)</td>
<td>batterystate (0-100) tempered (1)</td>
</tr>
<tr>
<td></td>
<td>Door/Window Sensor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Door/Window Sensor</td>
<td>Schlage*</td>
<td>Z-Wave</td>
<td>value (0/1) batterystate (0-100)</td>
<td></td>
<td>value_notify (0/1)</td>
<td>tempered (1)</td>
</tr>
<tr>
<td>Multi Sensor</td>
<td>AEON labs Multi Sensor</td>
<td>Z-Wave</td>
<td>value (0/1) humidity luminance temperature batterystate (0-100)</td>
<td></td>
<td></td>
<td>value_notify (0/1)</td>
</tr>
<tr>
<td>Lock</td>
<td>Schlage</td>
<td>Z-Wave</td>
<td>batterystate (0-100) lockState(0/1)</td>
<td>lockState (0/1) &quot;Locked&quot; or &quot;Unlocked&quot;</td>
<td>lockState(0/1) &quot;Locked&quot; or &quot;Unlocked&quot;</td>
<td></td>
</tr>
<tr>
<td>Lock</td>
<td>Yale*</td>
<td>Z-Wave</td>
<td>batterystate (0-100) lockState(0/1)</td>
<td>lockState (0/1) &quot;Locked&quot; or &quot;Unlocked&quot;</td>
<td>lockState(0/1) &quot;Locked&quot; or &quot;Unlocked&quot;</td>
<td></td>
</tr>
<tr>
<td>Electrical Outlet</td>
<td>GE*</td>
<td>Z-Wave</td>
<td>switchstate (0/255)</td>
<td>switchstate (0/1s)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Water Sensor</td>
<td>AEOTEC*</td>
<td>Z-Wave</td>
<td>value (0/1) batterystate (0-100)</td>
<td></td>
<td>value_notify (0/1)</td>
<td></td>
</tr>
<tr>
<td>Door/Window Sensor</td>
<td>ECOLINK*</td>
<td>Z-Wave</td>
<td>value (0/1) batterystate (0-100)</td>
<td></td>
<td>value_notify (0/1)</td>
<td>tempered (1)</td>
</tr>
<tr>
<td>Door/Window Sensor</td>
<td>EVERSPrING*</td>
<td>Z-Wave</td>
<td>value (0/1) batterystate (0-100)</td>
<td></td>
<td>value_notify (0/1)</td>
<td>batterystate (0-100)</td>
</tr>
<tr>
<td>Motion Sensor</td>
<td>ECOLINK</td>
<td>Z-Wave</td>
<td>batterystate(0-100)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smoke Sensor</td>
<td>First Alert*</td>
<td>Z-Wave</td>
<td>batterystate(0-100)</td>
<td></td>
<td></td>
<td>smokeAlert(0/1) batterystate</td>
</tr>
<tr>
<td>Device Type</td>
<td>Manufacturer</td>
<td>Protocol</td>
<td>Properties</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------</td>
<td>----------</td>
<td>------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Water Sensor</td>
<td>FortrezZ*</td>
<td>Z-Wave</td>
<td>batterystate (0-100)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dimmer Switch</td>
<td>COOPER*</td>
<td>Z-Wave</td>
<td>dimmingsetting(0-100)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dimmer Switch</td>
<td>GE*</td>
<td>Z-Wave</td>
<td>dimmingsetting(0-100)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dimmer Switch</td>
<td>Leviton*</td>
<td>Z-Wave</td>
<td>dimmingsetting(0-100)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Electrical Outlet</td>
<td>Leviton</td>
<td>Z-Wave</td>
<td>switchstate (0/255)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Door Window Sensor</td>
<td>Vision*</td>
<td>Z-Wave</td>
<td>batterystate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vibration and Shock Sensor</td>
<td>Vision</td>
<td>Z-Wave</td>
<td>Batterystate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Micro Smart Switch</td>
<td>Aeon Labs</td>
<td>Z-Wave</td>
<td>switchstate(0/1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Micro Smart Dimmer Switch</td>
<td>Aeon Labs</td>
<td>Z-Wave</td>
<td>dimmingsetting(0-100% or 255)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single Wall Smart Switch</td>
<td>TKBHome*</td>
<td>Z-Wave</td>
<td>switchstate(0/1), power(0-10)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Supported Devices

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Manufacturer</th>
<th>Protocol</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dual Wall Smart Switch</td>
<td>TKBHome</td>
<td>Z-Wave</td>
<td>switchstate(0/1), power(0-10)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>switchstate(0/1), power(0-10)</td>
</tr>
<tr>
<td>Dual Wall Dimmer Switch</td>
<td>TKBHome</td>
<td>Z-Wave</td>
<td>dimmingsetting(0-100% or 255)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>dimmingsetting(0-100% or 255)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>power(0-10)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>power(0-10)</td>
</tr>
<tr>
<td>Power Reader</td>
<td>NorthQ*</td>
<td>Z-Wave</td>
<td>batterystate</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>electricPowerKWH</td>
</tr>
<tr>
<td>Energy Meter</td>
<td>Aeon*</td>
<td>Z-Wave</td>
<td>batterystate</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>electricPowerKWH</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>electricPowerV</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>electricPowerW</td>
</tr>
<tr>
<td>Outlet</td>
<td>Aeon</td>
<td>Z-Wave</td>
<td>switchstate(0/1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>electricPowerV</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>electricPowerW</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>value_notify (0/1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>batterystate (0-100)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>tempered</td>
</tr>
<tr>
<td>2nc Edition D/W Sensor</td>
<td>Aeon</td>
<td>Z-Wave</td>
<td>value_notify (0/1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>batterystate (0-100)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>tempered</td>
</tr>
<tr>
<td>Light</td>
<td>Linear*</td>
<td>Z-Wave</td>
<td>dimmingsetting(0-99)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>dimmingsetting(0-99)</td>
</tr>
<tr>
<td>Multi-Sensor</td>
<td>Philio*</td>
<td>Z-Wave</td>
<td>Value_notify</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>temperature</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>luminance</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>batterystate</td>
</tr>
<tr>
<td>Radiator Thermostat</td>
<td>Danfoss Living Connect*</td>
<td>Z-Wave</td>
<td>batterystate (0-100)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>locProtectionState</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>heatingSetPoint</td>
</tr>
<tr>
<td>Philips Hue Kits</td>
<td>Philips* 456210</td>
<td>Wi-Fi / IP</td>
<td>value (0/1)</td>
</tr>
<tr>
<td></td>
<td>Hue A19 Bulb Starter Kit 2nd Generation</td>
<td></td>
<td>brightness(0-254)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>hue(0-65535)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>saturation(0-254)</td>
</tr>
<tr>
<td>Lock</td>
<td>Yale</td>
<td>ZigBee*</td>
<td>lockState(&quot;Locked&quot;/&quot;Unlocked&quot;)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ZCLVersion(0/255)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>powerSource</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>secondarySourcePresent(0/1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>identifying(0/1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>lockState(&quot;Locked&quot;/&quot;Unlocked&quot;)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>identifyTimeout(0/255)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>lockState(&quot;Locked&quot;/&quot;Unlocked&quot;)</td>
</tr>
</tbody>
</table>
## Supported Devices

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Manufacturer</th>
<th>Protocol</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motion Sensor</td>
<td>Centralite*</td>
<td>ZigBee</td>
<td>value1(0/1) deviceFailure(0/1) mainsFault(0/1) tampered(0/1) lowBattery(0/1) measuredTemperatureValue(-27315 to 32767) MinMeasuredTempValue(-27315 to 32767) MaxMeasuredTempValue(-27315 to 32767) ZCLVersion(0/255) powerSource SecondarySourcePresent(0/1) identifying(0/1)</td>
</tr>
<tr>
<td>Motion Sensor</td>
<td>Nyce*</td>
<td>ZigBee</td>
<td>value1(0/1) deviceFailure(0/1) mainsFault(0/1) tampered(0/1) lowBattery(1/1) measuredTemperatureValue(-27315 to 32767) MinMeasuredTempValue(-27315 to 32767) MaxMeasuredTempValue(-27315 to 32767) ZCLVersion(0/255) powerSource SecondarySourcePresent(0/1) identifying(0/1) humidity(0-100) minHumidity(0-99) maxHumidity(1-100)</td>
</tr>
<tr>
<td>Thermostat</td>
<td>ZEN Thermostat*</td>
<td>ZigBee</td>
<td>ZCLVersion(0-255) powerSource secondarySourcePresent(0/1) identifying(0/1) localTemperature(-27315 to 32767) range (for speed) identifyTimeout(0-65535) coolingSetPoint(-27315 to 32767) heatingSetPoint(-27315 to 32767) Speed(0-6) localTemperature(-27315 to 32767) Speed(0-6) coolingSetPoint(-27315 to 32767) heatingSetPoint(-27315 to 32767)</td>
</tr>
</tbody>
</table>
## Supported Devices

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Manufacturer</th>
<th>Protocol</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Water leak Sensor</td>
<td>Centralite</td>
<td>ZigBee</td>
<td>value1(0/1) value2(0/1) deviceFailure(0/1) mainsFault(0/1) tampered(0/1) lowBattery(0/1) measuredTemperature Value(-27315 to 32767) Min Measured Temp Value(-27315 to 32767) Max Measured Temp Value(-27315 to 32767) ZCLVersion(0-255) powerSource SecondarySourcePresent(0/1) identifying(0/1) identifyTimeout(0-65535) value1(0/1) value2(0/1) tampered(0/1) measuredTemperature Value(-27315 to 32767)</td>
</tr>
<tr>
<td>Lights</td>
<td>Philips Hue Light</td>
<td>ZigBee</td>
<td>currentLevel(0-255) ZCLVersion(0-255) powerSource SecondarySourcePresent(0/1) identifying(0/1) moveToLevel(0-255) levelTransitionTime (0-65535) moveMode(0-1) moveRate(0-255) moveToLevelWithOnOff(0-255) moveModeWithOnOff(0-1) stop value (0/1) hue (0-254) saturation(0-254) identifyTimeout(0-65535)</td>
</tr>
<tr>
<td>Lights</td>
<td>Philips Hue LED Strip</td>
<td>ZigBee</td>
<td>currentLevel(0-255) ZCLVersion(0-255) powerSource SecondarySourcePresent(0/1) identifying(0/1) moveToLevel(0-255) levelTransitionTime (0-65535) moveMode(0-1) moveRate(0-255) moveToLevelWithOnOff(0-255) moveModeWithOnOff(0-1) value (0/1) hue (0-254) saturation(0-254) identifyTimeout(0-65535)</td>
</tr>
<tr>
<td>Device Type</td>
<td>Manufacturer</td>
<td>Protocol</td>
<td>Properties</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Lights       | OSRAM® LED   | ZigBee   | currentLevel(0-255)  
ZCLVersion(0-255)  
powerSource(0/1)  
SecondarySourcePresent(0/1)  
identifying(0/1)  
value(0/1)  
moveToLevel(0-255)  
levelTransitionTime(0-65535)  
movemode(0-1)  
moverate(0-255)  
movetolevelwithonoff(0-255)  
movemodewithonoff(0-1)  
identifyTimeout(0-65535) |
| Lights       | GE®          | ZigBee   | currentLevel(0-255)  
ZCLVersion(0-255)  
powerSource(0/1)  
SecondarySourcePresent(0/1)  
identifying(0/1)  
value(0/1)  
moveToLevel(0-255)  
levelTransitionTime(0-65535)  
movemode(0-1)  
moverate(0-255)  
movetolevelwithonoff(0-255)  
movemodewithonoff(0-1)  
stop(0-1)  
identifyTimeout(0-65535) |
| Door Sensor  | Centralite   | ZigBee   | value1(0/1)  
value2(0/1)  
deviceFailure(0/1)  
mainsFault(0/1)  
tampered(0/1)  
lowBattery(0/1)  
measuredTemperatureValue(-27315 to 32767)  
MinMeasuredTemperatureValue(-27315 to 32767)  
MaxMeasuredTemperatureValue(-27315 to 32767)  
ZCLVersion(0-255)  
powerSource(0/1)  
SecondarySourcePresent(0/1)  
identifying(0/1)  
identifyTimeout(0-65535)  
value1(0/1)  
value2(0/1)  
tampered(0/1)  
measuredTemperatureValue(-27315 to 32767) |
## Supported Devices

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Manufacturer</th>
<th>Protocol</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lights</td>
<td>Cree*</td>
<td>ZigBee</td>
<td>currentLevel(0-255) ZCLVersion(0-255) powerSource SecondarySourcePresent(0/1) identifying(0/1) moveToLevel(0-255) levelTransitionTime(0-65535) moveMode(0-1) moveRate(0-255) moveToLevelWithOnOff(0-255) moveModeWithOnOff(0-1) Stop value(0/1) brightness identifyTimeout(0-65535)</td>
</tr>
<tr>
<td>Smart outlet</td>
<td>Centralite</td>
<td>ZigBee</td>
<td>ZCLVersion(0-255) powerSource SecondarySourcePresent(0/1) identifying(0/1) acFrequency(0-65535) rmsCurrent(0-65535) rmsVoltage(0-65535) activePower(-32768 to 32767) acCurrentMultiplier(1-65535) acCurrentDivisor(1-65535) acPowerMultiplier(1-65535) acPowerDivisor(1-65535) measurementType(0x00000000 – 0xFFFFFFFF)[32bit bitmap] value(0/1) IdentifyTimeout(0-65535)</td>
</tr>
<tr>
<td>Smart Switch</td>
<td>Quirky-GE*</td>
<td>ZigBee</td>
<td>ZCLVersion(0-255) powerSource SecondarySourcePresent(0/1) identifying(0/1) value(0/1) IdentifyTimeout(0-65535)</td>
</tr>
<tr>
<td>Smart outlet</td>
<td>GE*</td>
<td>ZigBee</td>
<td>ZCLVersion(0-255) powerSource SecondarySourcePresent(0/1) identifying(0/1) value(0/1) IdentifyTimeout(0-65535) value(0/1)</td>
</tr>
<tr>
<td>Device Type</td>
<td>Manufacturer</td>
<td>Protocol</td>
<td>Properties</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>--------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Temperature and humidity sensor</td>
<td>Centralite</td>
<td>ZigBee</td>
<td>ZCLVersion(0-255) powerSource secondarySourcePresent(0/1) identifying(0/1) measuredTemperatureValue(-27315 to 32767) MinMeasuredTempValue(-27315 to 32767) MaxMeasuredTempValue(-27315 to 32767) identifyTimeout(0-65535) measuredTemperatureValue(-27315 to 32767)</td>
</tr>
<tr>
<td>Thermostat</td>
<td>Centralite</td>
<td>ZigBee</td>
<td>ZCLVersion(0-255) powerSource secondarySourcePresent(0/1) identifying(0/1) localTemperature(-27315 to 32767) range (for speed) identifyTimeout(0-65535) coolingSetPoint(-27315 to 32767) heatingSetPoint(-27315 to 32767) Speed(0-6) localTemperature(-27315 to 32767) coolingSetPoint(-27315 to 32767) heatingSetPoint(-27315 to 32767)</td>
</tr>
<tr>
<td>Slim Multi Sensor</td>
<td>Philio</td>
<td>ZigBee</td>
<td>ZCLVersion(0-255) powerSource secondarySourcePresent(0/1) motion1(0/1) motion2(0/1) doorsense1(0/1) doorsense2(0/1) deviceFailure(0/1) mainsFault(0/1) tampered(0/1) lowBattery(0/1)</td>
</tr>
<tr>
<td>Flood Multi Sensor</td>
<td>Philio</td>
<td>ZigBee</td>
<td>value1(0/1) value2(0/1) deviceFailure(0/1) mainsFault(0/1) tampered(0/1) lowBattery(0/1) measuredTemperatureValue(-27315 to 32767) MinMeasuredTempValue(-27315 to 32767) MaxMeasuredTempValue(-27315 to 32767) humidity(0 to 100)</td>
</tr>
</tbody>
</table>
## Supported Devices

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Manufacturer</th>
<th>Protocol</th>
<th>Read-only</th>
<th>Read-Write</th>
<th>Notify</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi Sound Siren</td>
<td>Philio</td>
<td>ZigBee</td>
<td>minHumidity(0 to 99)</td>
<td>maxHumidity(1 to 100)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Alarm1(0/1)</td>
<td></td>
<td>maxDuration(0 to 65535)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Alarm2(0/1)</td>
<td></td>
<td>value1(0/1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>tampered(0/1)</td>
<td></td>
<td>value2(0/1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>lowBattery(0/1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smart Energy Plug</td>
<td>Philio</td>
<td>ZigBee</td>
<td>ZCLVersion(0-255)</td>
<td></td>
<td>value(0/1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>powerSource</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SecondarySourcePresent(0/1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>identifying(0/1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>acFrequency(0-65535)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>rmsCurrent(0-65535)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>rmsVoltage(0-65535)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>activePower(-32768 to 32767)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>acCurrentMultiplier(1-65535)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>acCurrentDivisor(1-65535)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>acPowerMultiplier(1-65535)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>acPowerDivisor(1-65535)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Note to Table 10:

1. The properties moveToLevel, levelTransitionTime, moveMode, moveRate, moveToLevelWithOnOff, moveModeWithOnOff, and Stop are write only. As they correspond to ZigBee* commands going to the end device, these properties cannot be read.

2. The meanings of Value1 and Value2 for different devices are as follows:

### Table 11. Meanings of Value1 and Value2 for Different Devices

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Value1/motion1/doorsesense1</th>
<th>Value2/motion2/doorsesense2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard CIE System</td>
<td>Alarm</td>
<td>-</td>
</tr>
<tr>
<td>Motion sensor</td>
<td>Intrusion indication</td>
<td>Presence indication</td>
</tr>
<tr>
<td>Contact switch / Door Sensor</td>
<td>1st portal Open/Close</td>
<td>2nd portal Open/Close</td>
</tr>
<tr>
<td>Fire sensor</td>
<td>Fire indication</td>
<td>-</td>
</tr>
</tbody>
</table>
### Supported Devices

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Value1/motion1/doorse1</th>
<th>Value2/motion2/doorse2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Water leak sensor</td>
<td>Water overflow indication</td>
<td>-</td>
</tr>
<tr>
<td>Gas sensor</td>
<td>CO indication</td>
<td>Cooking indication</td>
</tr>
<tr>
<td>Personal emergency device</td>
<td>Fall / Concussion</td>
<td>Emergency button</td>
</tr>
<tr>
<td>Vibration / Movement sensor</td>
<td>Movement indication</td>
<td>Vibration</td>
</tr>
<tr>
<td>Remote Control / Key fob / Keypad</td>
<td>Panic</td>
<td>Emergency</td>
</tr>
</tbody>
</table>

§
11.0 Troubleshooting Guide

11.1 Z-Wave* Stack

If the Z-Wave* stack is not working as expected, follow this section's steps to reset the Z-Wave stack.

11.1.1 Using Z-Wave* Sample Application

1. Copy the demos folder from the Z-Wave Module build system to the Gateway device. It can be placed at the /home/root/Intel directory created during the Intel SHDAP SDK installation. This folder could be located at (assuming the Sigma Z-Wave stack is built using Intel_SHDAP_SDK_Enable_ZWAVE_Module_Pre_Release_0_82):

   <path-to-ostro-os>/ostro-build/tmp-glibc/work/i586-nlp-32-ostro-linux/zipct1-sigma/1.0-r0/zipct1-sigma/demos

2. At the Gateway device, go to the demos folder and set the library path as follows:

   $ cd /home/root/Intel/demos
   $ export LD_LIBRARY_PATH=/home/shdap/Intel/shdap_sdk/bin

   This is the path where libzip_api.so and libzip_ctl.so are set up in the Z-Wave Interface Guide, Section 6.2 (Step 4), Document Number 567465.

3. Follow these steps to stop all Intel SHDAP services, including "shdap-root" and start "zipgateway" manually:

   $ systemctl stop shdap.service
   $ systemctl stop broker.service
   $ systemctl stop shdap-root.service
   $ systemctl stop serial-getty@ttyS1.service
   $ iptables --policy INPUT ACCEPT

   $ /usr/sbin/udhcpd /usr/local/etc/udhcpd.conf
   $ /usr/bin/zipgateway --e /usr/local/etc/zipgateway.cfg

4. Find the IP address of Zipgateway: Go to demos/gw_discovery and run the application.

   ./gw_discovery

   The application will print the IP address assigned to the zipgateway on the console. Note the IP address.
5. Go to demos/nw_reset and update the following variables in app.cfg before running the application.
   Set the value.
   ZipRouterIP = <ip address noted in above step>
   DTLSPSK = 12345678901234567890123456789012

6. Run the nw_reset application.
   $ chmod +x nw_reset
   $ ./nw_reset

   This application resets the Z-Wave stack.

7. Soft-reboot the device using the reboot command and check whether the Z-Wave functionality is working as expected.

8. Other demo applications from Sigma Designs can be tested, namely add_node, rm_node, basic, and bin_switch, which are available in the demos folder. In these demo applications, the following steps can be taken as a reference. Here add_node is used as an example. add_node and rm_node can be used to pair and unpair the Z-Wave sensor device with the Z-Wave network, respectively. If able to add and remove a device using this application, the Z-Wave sensor device is working properly.
   a. Go to demos/add_node folder.
   b. Update the app.cfg configuration file as described in Step 5.
   c. Ensure that the LD_LIBRARY_PATH is set to the directory containing libzip_api.so and libzip_ctl.so.
      $ export LD_LIBRARY_PATH=/home/shdap/Intel/shdap_sdk/bin
   d. Run the add_node application network.
      Chmod +x add_node
      ./add_node
   e. Wait for the message “InitializationStatus: 0”.
   f. Select Option 1 to add a sensor device to the network.
   g. Put the sensor device in pairing mode. After some time, a message appears on the console, stating that the device is successfully added to the network.
   h. After adding the device to the network, remove the device properly using the rm_node application before shutting down the system.

11.2 Verify Bluetooth Interface on Intel® SHDAP Gateways

Following are recommended steps to verify Bluetooth or interface directly with Bluetooth on a gateway:

1. Ensure that the Bluetooth LE plugin is not enabled in the Intel® SHDAP SDK:
Troubleshooting Guide

a. Remove the line `libbleplugin.so` (Bluetooth LE plugin) if present in this file:
/home/shdap/Intel/shdap_sdk/shdap/bin/plugin_load_list.txt

b. If you edited the file in the previous step, do a soft reboot of the gateway.

2. Initialize the Bluetooth interface with this command:

   $ rfkill unblock bluetooth
   $ hciconfig hci0 up

3. Device Discovery, Non-Bluetooth LE: Run the following command:

   $ hcitool scan
   $ hcitool cc <MAC ID>
   $ l2ping -c 5 <MAC ID>

4. Device Discovery, Bluetooth LE: Initialize the Bluetooth interface:

   $ hcitool lescan
   $ gatttool -b <MAC ID> -I

Example log:

Note: You might need to run the lescan command twice if connection is not successful in gatttool.

root@intel-quark:~# hcitool lescan
LE Scan ...
02:18:0D:EE:4F:98 amulet-060
root@intel-quark:~# hcitool lescan
LE Scan ...
02:18:0D:EE:4F:98 amulet-060
^C
root@intel-quark:~# gatttool -b 02:18:0D:EE:4F:98 -I
[02:18:0D:EE:4F:98][LE]> connect
Attempting to connect to 02:18:0D:EE:4F:98
Connection successful
[02:18:0D:EE:4F:98][LE]> char-write-cmd 0x29 01(write data)