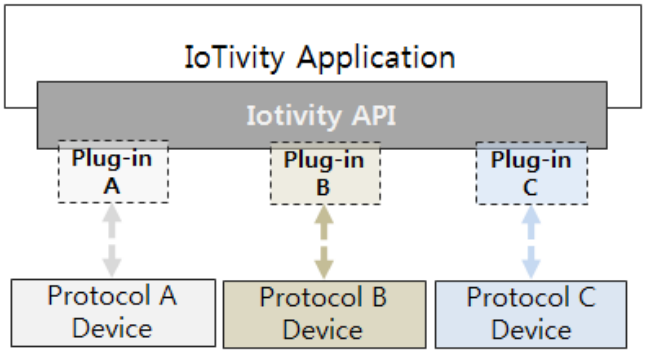
IoTivity Programmer's Guide
– Protocol Plug-in Manager for Android

1 CONTENTS

2 OVERVIEW

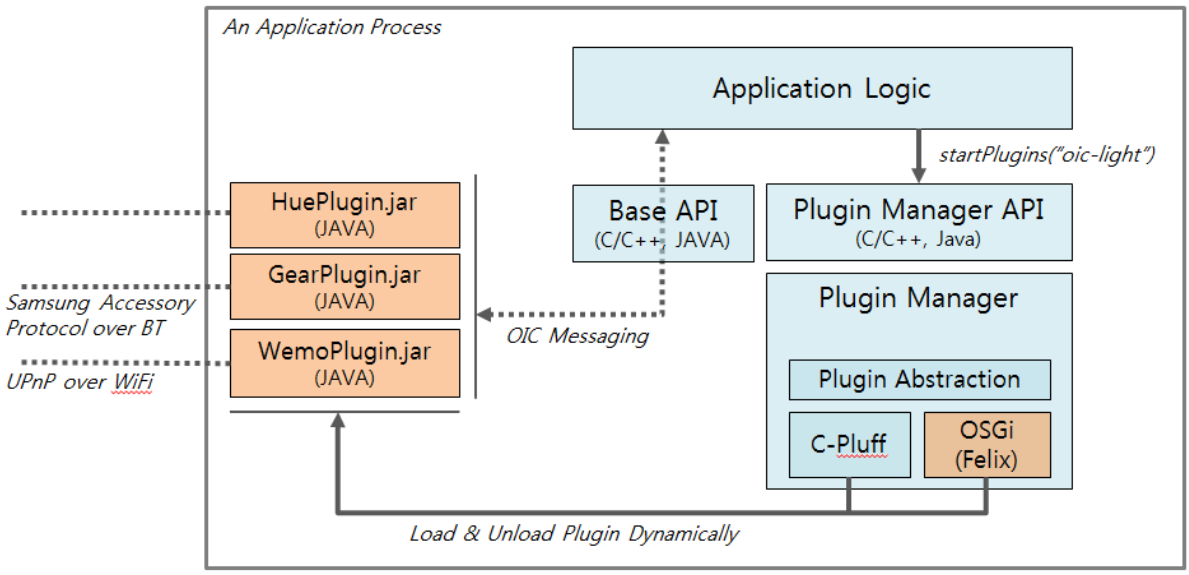
This guide will help you to use protocol plugins. Using protocol plugins, your application can communicate with various protocol devices using IoTivity APIs as shown in the following diagram.



<Figure 1. Protocol Plugin Concept>

2.1 OVERALL FLOWS

Using Plugin Manager APIs, application can start plugins that are located in a specific folder. After starting a plugin, the plugin will try to find its device using own protocol and creates resource server when the device is found. Then application can find and communicate with the resource using base APIs similar to normal IoTivity resource. Following diagram describes the flows.



<Figure 2. Overall Flow>

3 USING PLUGIN MANAGER

This section describes how to start plugins using plugin manager.

3.1 SETTING PLUGIN CONFIGURATION

For plugin configuration, *pluginmanager.xml* file should be located in the assests/files folder of Android application. Plugin manager will load the configuration information when the application creates plugin manager instance. By editing the configuration file, application developer can change plugins.

```
<?xml version="1.0" encoding="utf-8"?>
<pluginManager>
  <pluginInfo
    PluginPath="/data/org.iotivity.service.ppm/files">
    name="pluginmanager"
  </pluginInfo>
</pluginManager>
```

3.2 LOCATING PLUGIN AND MANIFEST FILE

Before starting plugins, plugin jar files which are generated using 3rd party jars should be located in the PluginPath specified in the *pluginmanager.xml*.

Note: For detailed information on how to generate plugin jars using the 3rd party jars for each of the plugins, please refer “Getting Started - Protocol Plugin Manager for Android”

Following is the folder structure containing 3rd party jar files information.

```
/plugins/Android/
        plugin.gear.noti
        plugin.hue
        plugin.wemo
/plugin.gear.noti
    lib
        gson.jar
        richnotification.jar
        sdk.jar
   META-INF
        MANIFEST.MF

/plugin.hue
    lib
        huelocalsdk.jar
        huesdkresources.jar
    META-INF
        MANIFEST.MF

/plugin.wemo
    lib
        wemosdk.jar
   META-INF
        MANIFEST.MF
```

Each plugin has a manifest file (MANIFEST.MF) in its folder and will have following information.

Key Name	**Description**
Manifest-Version	Version of the Manifest file
Bundle-Name	Name of the plugin
Bundle-SymbolicName	Symbolic name of the plugin
Bundle-ResourceType	Supported OIC resource type by the plugin
Bundle-Version	Version of the plugin
Bundle-ClassPath	.classpath file path for the plugin
Export-Package	List of plugin packages
Import-Package	Packages required by the plugin
Bundle-Activator	Plugin Activator package path. (Example: oic.plugin.gear.noti.Activator)

Following is the description of **Philips Hue Plugin's** manifest file.

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Philips Hue Plugin
Bundle-SymbolicName: hue
Bundle-ResourceType: device.light
Bundle-Version: 1.0.0
Bundle-ClassPath: .,
 libs/
Export-Package: com.philips.lighting.annotations,
 com.philips.lighting.hue.listener,
 com.philips.lighting.hue.sdk,
 com.philips.lighting.hue.sdk.bridge.impl,
 ……
Import-Package: org.osgi.framework,
 android.app,
 android.dalvik;resolution:=optional,
 android.util,
 android.content,
……
Bundle-Activator: oic.plugin.hue.Activator
```

4 USING PLUGIN RESOURCES

This section describes how to communicate with non-oic devices using plugins and IoTivity API.

4.1 PHILPS HUE PLUGIN

Application can find Hue device with “device.light” resource type and communicate with following attributes.

Attribute Key	Attribute Value	Type	Description
power	“on”, “off”	String	Turn on/off Hue bulb
color	0~10	Integer	Change color of the bulb

4.2 SAMSUNG GALAXY GEAR S RICH NOTIFICATION PLUGIN

Application can find Galaxy Gear S device using “device.notify” resource type and communicate with following attributes

Attribute Key	Attribute Value	Type	Description
notify	text	String	Send text notification to Galaxy Gear S

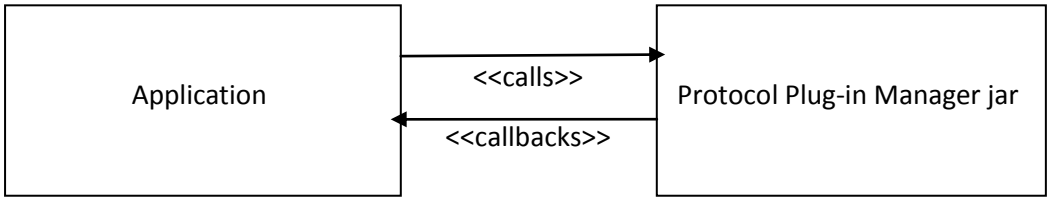
4.3 BELKIN WEMO PLUGIN

Application can find Wemo device using “device.smartplug” resource type and communicate with following attributes

Attribute Key	Attribute Value	Type	Description
power	“on”, “off”	String	Turn on/off the Wemo switch

5 ANDROID SDK API

This section provides information on the APIs exposed by Protocol Plug-in Manager service for the use by applications. SDK API is the facet of Protocol Plug-in Manager to applications as shown in the Figure 3.



<Figure 3. Protocol Plug-in Manager SDK APIs and Application >

5.1 PROTOCOL PLUG-IN MANAGER API

These APIs provide methods for application to start and stop the plug-ins, scan for plug-ins in the registered directory, get the list of plug-ins and also the state of plug-ins. The operations provided in the SDK are listed below:

- startPlugins
- stopPlugins
- rescanPlugin
- getPlugins
- getState

startPlugins API can be used to start the plugins by specifying key and value as parameters. Using the plugin information described in the manifest file, application can start plugins as follows.

startPlugins(“resourcetype”, “device.smartplug”);

startPlugins(“id”, “wemo”);

After starting, the plugin will try to find its device using its own protocol and will create a resource server when the device is found. Then the application can find and communicate with the resource using the base API as a normal IoTivity resource.

Prototype:

- int startPlugins(String key, String value);

Parmaters:

- key - Key string of the plug-in to be started.
- value - Value string of the plug-in to be started.

Return Value:

- Returns 1 on Success, 0 on Failure.

stopPlugins API can be used to stop the plugins by specifying key and value as parameters. Key can be name of a resource type (Example: ResourceType) and value is the resource type value (Example: device.light). Once this API is called, the application can no longer find and communicate with the resource.

Prototype:

- int stopPlugins(String key, String value)

Parmaters:

- key - Key string of the plug-in to be stopped.
- value - Value string of the plug-in to be stopped.

Return Value:

- Returns 1 on Success, 0 on Failure.

rescanPlugin API can be used to rescan for plug-ins in the registered directory and to install those plug-ins in the plug-in manager table.

Prototype:

- int rescanPlugin ();

Return Value:

Returns 1 on Success, 0 on Failure.

getPlugins API can be used to get the list of Plug-ins that are installed. An application can get the information of plugin as folllows.

Vector<Plugin> plugins = getPlugins();

Prototype:

- Plugin[] getPlugins()

Return Value:

Returns available plug-ins' information in an Array.

getState API can be used to get the state of the plug-in by providing plug-in ID as parameter. This API returns the plug-in state in a string.

Prototype:

- String getState(String plugID)

Parmaters:

- plugID - ID of the plug-in for which state is being queried.

Return Value:

- Returns the state of the plug-in in a String.

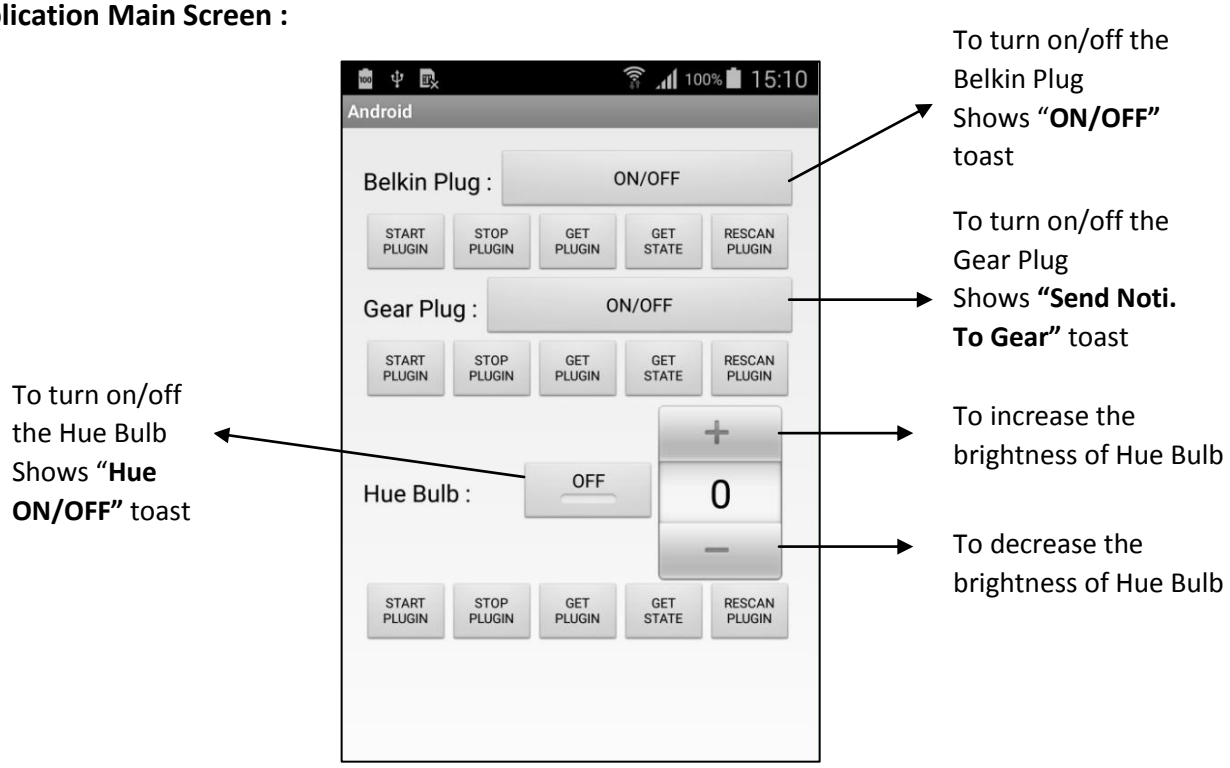
6 EXAMPLE

This section describes Sample Application for Protocol plugin manager.

6.1 ANDROID SAMPLE APPLICATION

This section shows the actions that we can perform on Belkin Plug, Gear Plug and Hue Bulb devices using the plugins available in the assets folder of the application. Please refer Using Plugin Manager section for information on plugin jar files.

Application Main Screen :



6.1.1 START PLUGIN

Starts the plugin by taking its ResourceType as parameter.

For Belkin Plug, the ResourceType is "**device.smartplug**".

For Gear Plug, the ResourceType is "**device.notify**".

For Belkin Plug, the ResourceType is "**device.light**".

After starting the plugin, our application finds those devices and reads their properties using the Base APIs. Then it shows a Toast indicating that the device is connected. For Ex: "**Belkin Connected**".

Start Plugin has to be called first to perform the actions on the devices such as turn ON/OFF.

6.1.2 STOP PLUGIN

Stops the plugin by taking its ResourceType as parameter.

To find and perform actions on the devices after stopping the plugin, start plugin has to be called.

6.1.3 GET PLUGIN

It returns an array of Plugin objects for the plugins located in the assets folder of the application.

Using the plugin objects, we can obtain the ID, Name, Version, and ProviderName of the plugins using the Plugin APIs.

6.1.4 GET STATE

To get the current state of the plugin by providing the Plugin-ID as parameter.

The Plugin State can be any of the following:

- **INSTALLED**: Before starting the plugin
- **ACTIVE**: After starting the plugin
- **RESOLVED**: After stopping the plugin

6.1.5 RESCAN PLUGIN

It re-scans for all the plugins in the plugin path (See Using Plugin Manager).