Iotivity Programmer's Guide – Protocol Plugin Usage Guide

1 CONTENTS

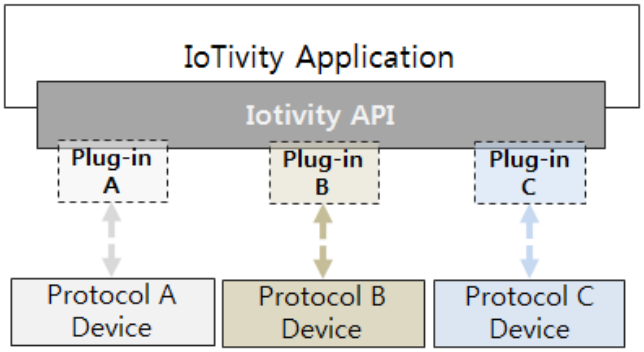
2 Revision History 3
3 Overview 4
3.1 Overall Flows 4
4 Using Plugin Manager 5
4.1 Setting Plugin Configuration 5
4.2 Locating Plugin and Menifest File 5
4.3 Starting Plugins with Attribute 6
4.4 Showing Plugin Information 6
5 Using Plugin Resources 7
5.1 HUE Plugin 7
5.2 Reference MQTT Fan Plugin 7

2 REVISION HISTORY

Revision	Date	Author(s)	Comments
v0.1	17/12/2014	KC Park	Initial Preview Release

3 OVERVIEW

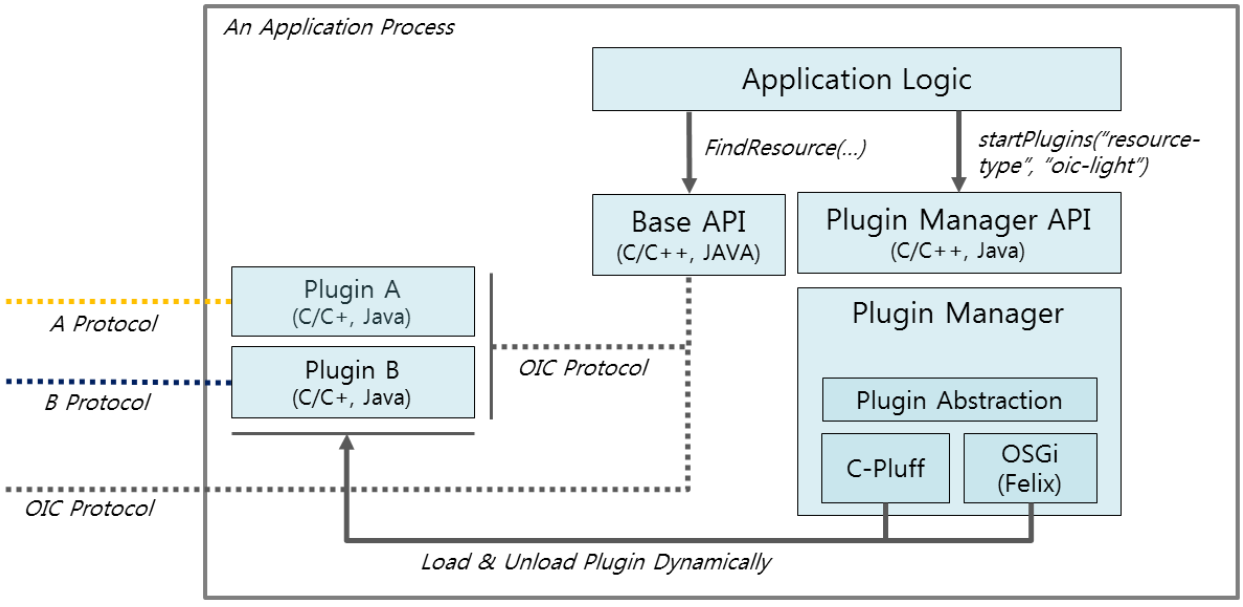
This guide will help you to use protocol plugins. Using protocol plugins, your application can communicate with various protocol devices using IoTivity API as following diagram.



<Figure 1. Protocol Plugin Concept>

3.1 OVERALL FLOWS

Using Plugin Manager API, application can start plugins which located in a specific folder. After starting a plugin, the plugin will try to find it's device using own protocol and create resource server when the device is found. Then application can find and communicate with the resource using base api as same as normal IoTivity resource. Following diagram describes the flows.



<Figure 2. Overall Flow>

4 USING PLUGIN MANAGER

This guide is about how to start plugins using plugin manager.

4.1 SETTING PLUGIN CONFIGURATION

For plugin configuration, PluginManager.xml file should be located in the folder which application executable file exists. Then, plugin manager can load the config information when application creates plugin manager instance. By editing the configuration file, application developer can change plugins folder, max number of plugins in theas following.

```
<?xml version="1.0" encoding="utf-8"?>
<pluginManager>
  <pluginInfo
    PluginPath="./plugins"
    MaxPlugin="50" >
  </pluginInfo>
```

4.2 LOCATING PLUGIN AND MENIFEST FILE

Before starting plugins, you should locate plugin binary into in the path specified in the plugin configuration and plugins should be in the path with separated folder.

```
/sample-app
  - sample-executable
  - pluginmanager.xml
  - libpmimpl.so
/ sample-app /plugins
/ sample-app /plugins/mqtt-fan
   - mqttfanplugin.so
   - plugin.xml
/ sample-app /plugins/hue
   - hueplugin.so
   - plugin.xml
```

Each plugin have manifest XML file describing it's information in the same folder and will have following information.

Key Name	Description
id	Unique id of the plugin
version	Version of the plugin
name	Name of the plugin
resourcetype	Supported OIC resource type by the plugin
provider-name	Provider name of the plugin

Following XML description is an plugin manifest file about Philips Hue Plugin.

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
  id="oic.plugin.hue"
  version="0.1"
  name="hue plugin"
  resourcetype=”oic.light”
  provider-name="wallace">
  <runtime library="libplugin-hue-light" funcs="hue_light"/>
</plugin>
```

4.3 STARTING PLUGINS WITH ATTRIBUTE

With plugin information which described in the manifest file, application can start plugins.

m_pm->startPlugins(“resourcetype”, “oic.light”);

m_pm->startPlugins(“id”, “oic.plugin.hue”);

4.4 SHOWING PLUGIN INFORMATION

After creating plugin manager instance, application can get information of plugin as folllows.

PluginManager *m_pm = new PluginManager();

std::vector<Plugin> plugins = m_pm->getPlugins();

```
std::string name = plugins[0].getName();

std::string id = plugins[0].getId();
```

5 USING PLUGIN RESOURCES

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

5.1 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
brightness	0~125	Integer	Change brightness of the bulb
color	0~65,535	Integer	Change color of the bulb

5.2 MQTT FAN PLUGIN

Application can find MQTT FAN device using “oic.fan” resource type and communicate with following attributes

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