IoTivity Programmer’s Guide
– Protocol Plugin Manager for Linux

1 CONTENTS

2 Overview 3

2.1 Overall Flows 3

3 Using Plugin Manager 4

3.1 Setting Plugin Configuration 4

3.2 Locating Plugin and Manifest File 4

3.3 Starting Plugins with Attribute 5

3.4 Getting Plugin Information 6

4 Using Plugin Resources 6

4.1 MQTT Fan Plugin 6

5 SDK API 6

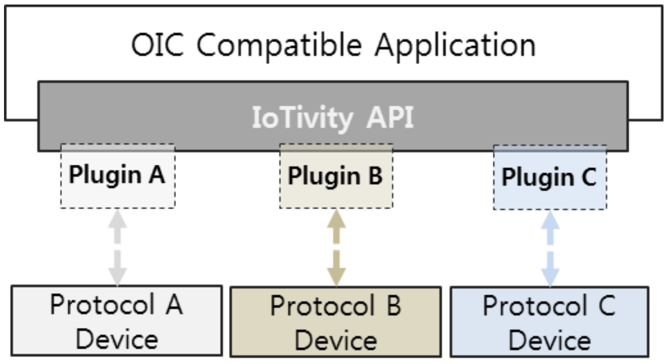
5.1 Protocol Plugin Manager API 7

6 Example 9

6.1 Linux Sample Application 9

2 OVERVIEW

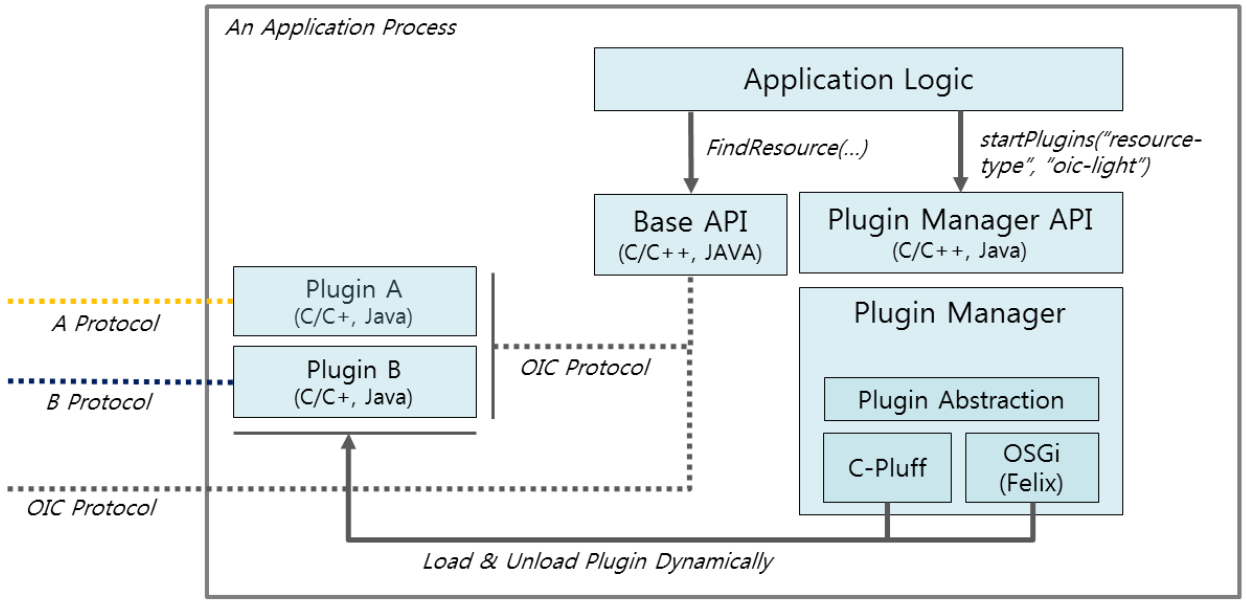
This guide will help you to use protocol plugins and plugin manager. Using protocol plugins, your application wii be able to communicate with various heterogeneous protocol devices using IoTivity API as presented in the following diagram.



<Figure 1. Protocol Plugin Concept>

2.1 OVERALL FLOWS

Using Plugin Manager API, application can start plugins located in a specific folder. After starting a plugin, the plugin will try to find devices using its own protocol and creates resource server when a device is found. Then the application can find and communicate with the resource using base API in the same manner as a normal IoTivity resource. The following diagram describes the corresponding flows.



<Figure 2. Overall Flow>

3 Using Plugin Manager

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

3.1 Setting Plugin Configuration

For plugin configuration, *pluginmanager.xml* file should be located in the folder in which the 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.

```
<?xml version="1.0" encoding="utf-8"?>

<pluginManager>

   <pluginInfo

     PluginPath="./plugins">

   </pluginInfo>

</pluginManager>
```

3.2 Locating Plugin and Manifest File

Before starting plugins, plugin binaries should be located in the path specified in the plugin configuration (e.g., *libpmimpl.so* located in the path /sample-app). In addition, each plugins should be located in the separate folder (e.g., /sample-app/plugins/mqtt-fan and /sample-app /plugins/hue as shown below).

```
/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 should have manifest XML file describing the following information and the manifest file should be located within the same folder as the plugin source code.

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

The following XML description is a plugin manifest file of Philips Hue Plugin.

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
  id="oic.plugin.mqtt-fan"
  version="0.1"
 name="mqtt-fan"
 resourcetype="oic.fan" >
  <runtime library="fanserver_mqtt_plugin" funcs="mqtt_plugin_fanserver_funcs"/>
</plugin>
```

3.3 Starting Plugins with Attribute

With plugin information described in the manifest XML file, application can start plugins using the following methods.

```
m_pm->startPlugins("resourcetype", "oic.fan");
m_pm->startPlugins("id", "oic.plugin.mqtt-fan");
```

3.4 Getting Plugin Information

After creating plugin manager instance, application can get information of the 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();
```

4 Using Plugin Resources

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

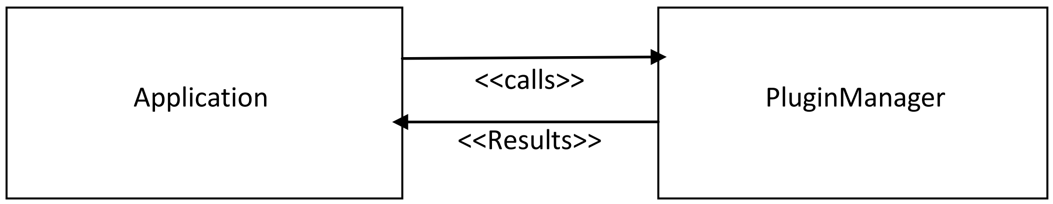
4.1 MQTT Fan Plugin

Application can find MQTT FAN device using "oic.fan" resource type and communicate with the following attribute.

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

5 SDK API

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



<Figure 3. Protocol Plugin Manager SDK APIs and Application >

5.1 Protocol Plugin Manager API

"Protocol Plugin Manager" APIs provide methods for application to start and stop the plugins, scan for plugins in the registered directory, get the list of plugins and also the state of plugins. 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", "oic.fan");

startPlugins("id", "oic.plugin.mqtt-fan");
```

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 PluginManager::startPlugins(const std::string key, const std::string value)
```

Parmaters:

- key - Key string of the plugin to be started.
- value - Value string of the plugin 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(const std::string key, const std::string value);

Parmaters:

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

Return Value:

- Returns 1 on Success, 0 on Failure.

rescanPlugin API can be used to rescan for plugins in the registered directory and to install those plugins in the plugin manager table.

Prototype:

int rescanPlugin();

Return Value:

- Returns 1 on Success, 0 on Failure.

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

Prototype:

std::vector<Plugin> getPlugins(void);

Return Value:

- Returns available plugins' information in an V.

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

Prototype:

std::string getState(const std::string plugID);

Parmaters:

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

Return Value:

- Returns the state of the plugin in a String.

6 EXAMPLE

This section describes the Sample Application used for Protocol plugin manager.

6.1 LINUX SAMPLE APPLICATION

This section describes flow of sample application where we try to start, find and to perform operation on the “mqtt-fan”plugin located in the plugins folder.

To start the application, run the **mqttclient** program as shown below.

```
~/iotivity/service/protocol-plugin/sample-app/linux/mqtt$ ./mqttclient

Current path is ../../../plugins

====== Plugins List ======

 ID                        NAME        STATE        TYPE

 oic.plugin.mqtt-fan       mqtt-fan     INSTALLED    oic.fan

 oic.plugin.mqtt-light     mqtt-light   INSTALLED    oic.light
```

Initially it creates an instance of PluginManager and shows the list of plugins available in the plugins folder whose path is specified in the **pluginmanager.xml**.

Then it starts the fan plugin by passing the ResourceType(**oic.fan**) to the StartPlugins() API of PluginManager.

After starting the plugin, the fan resource will be discovered by calling the findResource() API of OCPlatform. Once the fan resource is discovered, its URI, Host address, Resource Types and Resources Interfaces will be displayed as shown below.

```
 ~/iotivity/service/protocol-plugin/sample-app/linux/mqtt$ ./mqttclient

…………………………………..(Plugins List as shown above)…………………………………..

start plug-in oic.plugin.mqtt-fan.

start_fanserver [mosquitto] Null

Mosquitto is working

Mosquitto Connection is done

FanResource register time is: Mon Apr 20 12:26:37 2015

Finding Resource...
```

```
Finding Resource...

DISCOVERED Resource:

        URI of the resource: /a/fan

        Host address of the resource: coap://107.108.81.116:33279

        List of resource types:

                core.fan

        List of resource interfaces:

                oc.mi.def
```

After discovering the fan resource, application will send put request to change the power of the fan resource (1 - ON , 0 - OFF) as shown below.

```
Putting fan representation...
In entity handler wrapper:
    In Server CPP entity handler:
        requestFlag : Request   ===  Handle by FanServer
            requestType : PUT
                state: true
                power: 1
PUT request was successful
    state: true
    power: 1
    name: John's fan
```