Defining Thunder Service Plugins

RDK Services

RDK Services or Thunder Plugins are available in :

- Open source RDK Services are in rdkcentral github - https://github.com/rdkcentral/rdkservices/

repo url: https://github.com/rdkcentral/rdkservices.git

bitbake recipe: https://code.rdkcentral.com/r/plugins/gitiles/components/generic/rdk-oe/meta-rdk-video/+/refs/heads/rdk-next/recipes-extended/rdkservices/rdkservices_git.bb

Typical structure of project tree of thunder plugin

```
├── CMakeLists.txt
└── SimpleService
    ├── cmake
    │   ├── FindDS.cmake
    │   └── FindIARMBus.cmake
    ├── CMakeLists.txt
    ├── LICENSE
    ├── Module.cpp
    ├── Module.h
    ├── README.md
    ├── SimpleService.config
    ├── SimpleService.cpp
    ├── SimpleService.h
    └── SimpleService.json
```

Steps to add new Open Source RDK Service

RDKServices github repo will get checked out under /tmp/work folder and changes can be made there.

tmp/work/mips32el-rdk-linux/rdkservices/3.0+gitAUTOINC+fa49ddedd1-r1/git

Here is the description step by step to add new plugin with the name SimpleService based on the file structure of DisplaySettings. Display settings requires IARM and DeviceSettings libraries. Other services may not have these dependencies, depending upon their complexity.

1. copy the folder tmp/work/mips32el-rdk-linux/rdkservices/3.0+gitAUTOINC+fa49ddedd1-r1/git/DisplaySettings to SimpleService with full content;
2. in folder tmp/work/mips32el-rdk-linux/rdkservices/3.0+gitAUTOINC+fa49ddedd1-r1/git/SimpleService rename all files DisplaySettings to SimpleService, keeping the current extensions
3. Rename all cases of DisplaySettings to SimpleService in all files under SimpleService;
4. Refactor each of the plugin methods.
    a. getQuirks - this should be done for any plugin that defines it. Quirks are the mechanism to describe any low level bugs or changes that may have been fixed within a specific API version.
    b. API methods
    c. API events
    d. Do not refactor any methods not part of the API, with exception of getQuirks above. Note that the Javascript related methods were intended for use with Javascript clients through QtWebkit and do not need to be included in the Thunder service. getJavaScriptObject and <servicename>JavaScriptObject are examples of these.
5. Legacy services are using Qt library, thunder plugin should not use it. So the std::vector or std::list can be used instead of QList, std::thread instead of QThread, WPEFramework::Core::TimerType instead of QTimer.
6. Now everything is available to build the new plugin:
    - bitbake rdkservices
7. Copy files on stb: etc/WPEFramework/plugins/SimpleService.json and usr/lib/wpeframework/plugins/libWPEFrameworkSimpleService.so
8. Restart thunder and test new plugin:
    - systemctl restart wpeframework
    - curl -d '{"jsonrpc":"2.0","id":"3","method": "SimpleService.1.getQuirks"}' http://127.0.0.1:9998/jsonrpc
9. Result for these steps should be:

```
{"jsonrpc":"2.0","id":3,"result":{"quirks":["XRE-7389","DELIA-16415","RDK-16024","DELIA-18552"],"success":
true}}
```

Now SimpleService can be refactored to exclude all unused methods, events and additional code, which are no longer required.

Adding new versions for RDK Services

All RDK Services will start with version 1. If there are client-facing changes (adding or removing methods; adding, removing or changing parameters in methods; changing event names or payloads) then the version number of the service needs to be incremented. Here are the details on how to increment the version.

Versions are maintained in different JSONRPC handlers. We need to specify the list of versions as an array to each of these handlers.

Assuming we are adding a new API setFoo() to version 2 for DisplaySettings plugin. We need to create a new JSONRPC handler for this version. This basically takes an array of versions that are supported by this handler.

DisplaySettings.cpp

```
DisplaySettings::DisplaySettings()
: AbstractPlugin()
{...// Create JSONRPC handler for version 2. The last parameter makes sure that all handlers are copied from
the base version to this new one...
Core::JSONRPC::Handler& Version_2 = JSONRPC::CreateHandler({ 2 }, *this);

// Register the new method in version 2 to this handler.
Version_2.Register<Core::JSON::String>(_T("setFoo"), &DisplaySettings::setFoo, this);
}

//version 2 APIs
void DisplaySettings::setFoo(Core::JSON::String)
{
}
```