Release Notes

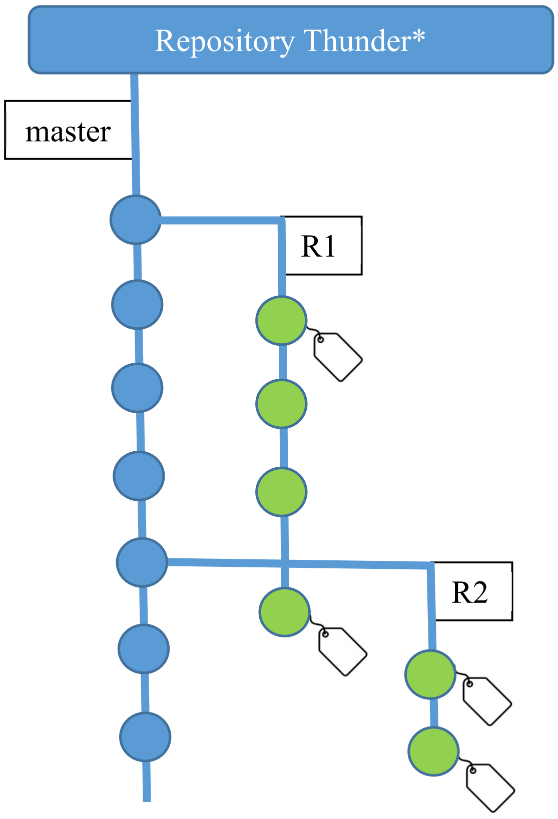
Thunder R3

Author	Version	Date
Marcel Fransen	0 .1 (Draft)	2022/7/18

1.Process description

The Thunder framework is released on branches. Any major release has its own branch (R1/R2/R_x_). This document will refer to these branches as “release branch”. The master branch contains future development and in due time will result in a new major release and thus a new RX branch (Release branch). As a rule of thumb, on major release branches no new functionality is added, only fixes to the Thunder Framework could be added to branch. New features are developed on the master branch, also referred to as the development branch.

A new release branch gets release notes describing the functionality extensions realized in the release branch (this document).



On the right a graphical representation of the repository layout is shown. The circles represent a commit. The lines going down represent the timeline of a branch.

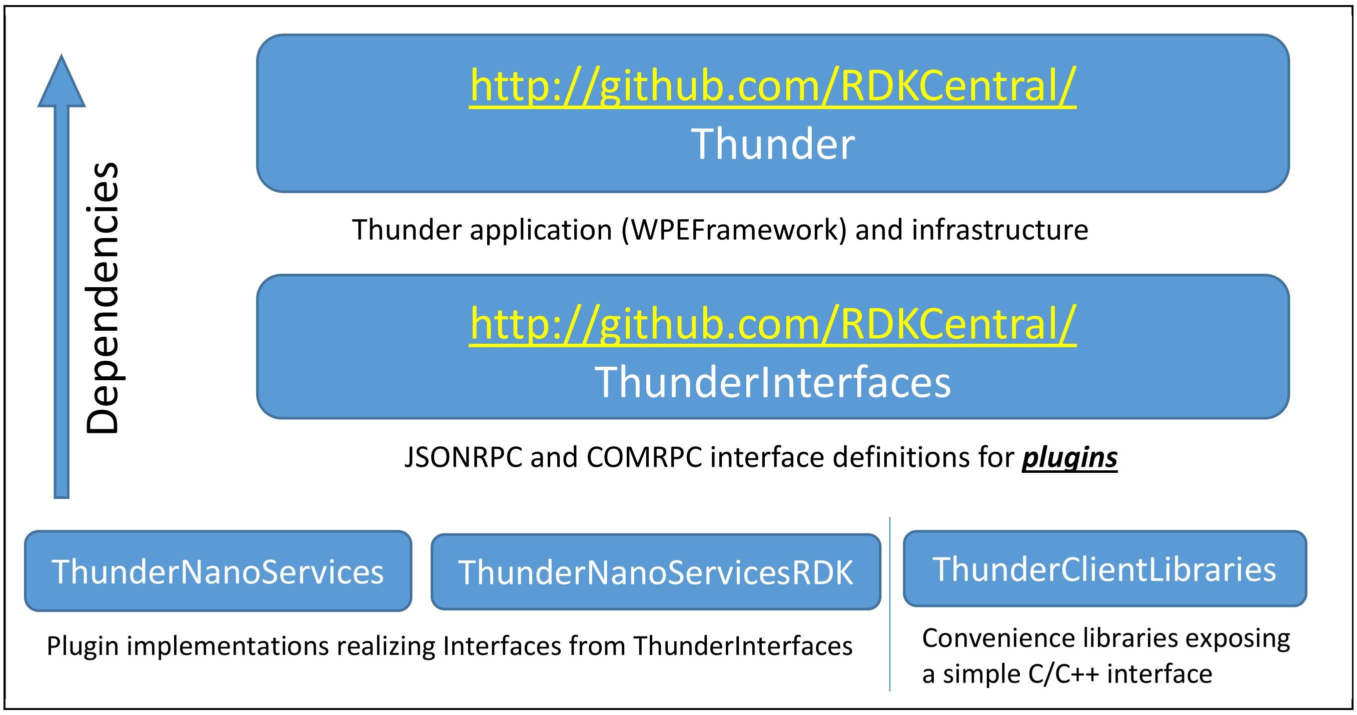
The first line going done is the master branch commits on this line related to future development. At a given point in time, a commit on this line is used to create a release. Indicated by the next vertical blue line down. On this branch, the commits seen, are only fixes to a reported issue. No active development takes place here.

Once the QA department has approved the commits, they will tag that commits used on that branch with a tag (typically R_x_-v_y_._z_). Operators are expected to only use release branches (Rx) and preferably only tagged commits on these release branches.

Release branches between the different repositories can not be mixed. All Thunder repositories used in a build should use master or same release branch available in all repositories. All Thunder builds must use *master*, for the development of new functionality within Thunder or the same *release* branch, in all repositories, for users of the framework and production deployment.

2.Packages

As of R2, the Thunder framework consists of 2 additional repositories (*ThunderInterfaces* and *ThunderClientLibraries*). Each repository has its own rationale. Here an overview:


Dependencies
http://github.com/RDKCentral/
Thunder
Thunder application (WPEFramework) and infrastructure
http://github.com/RDKCentral/
ThunderInterfaces
JSONRPC and COMRPC interface definitions for plugins
ThunderNanoServices
ThunderNanoServicesRDK
ThunderClientLibraries
Plugin implementations realizing Interfaces from ThunderInterfaces
Convenience libraries exposing a simple C/C++ interface

Repositories only have, by design, only an upward dependency. This means that repositories on the same vertical line are ***not*** dependent on each other. This means that none of the sources in *ThunderClientLibraries* is allowed to reference/use anything in the *ThunderNanoServices* or *ThunderNanoServicesRDK*.

- **Thunder:**

Dependencies: ***ZLIB***, optional: OpenSSL,LXC,RUNC,CRUN,Dobby.

This repository contains platform abstractions for different OS'es (Windows/Linux and Apple OS-X). Abstractions are available for Network handling (interface control and communication) Bluetooth (GATT/L2CAP/HCI), Containers (LXC/CRUN/RUNC/Dobby), Broadcast Tuner support, JSON parsing, Proceses, Threading, Synchronization.

This repository also contains functional blocks to be used by the plugins like:

Tracing/Logging:	Runtime enabling of traces, low intervention.
COMRPC:	Mechanism for C/C++ calls to be agnostic for process boundaries.
JSONRPC:	C++ support classes to be able to simply interact with JSONRPC services from the native world.
WebSockets:	C++ support class to work with the RFC for HTTP communication and web socket communication.
WPEFramework:	The Thunder application that controls the plugin instantiation and runtime states, in/out of process/in a container and routes JSONRPC/COMRPC traffic to the designated plugin.

• ThunderInterfaces:

Dependencies: ***Thunder***.

This repository holds all interfaces (JSONRPC and COMRPC) that are used across the plugins and or client libraries. These interfaces contain no implementations what-so-ever. The implementations should always reside in the plugin repositories. Third parties that would like to interface with one of the plugins (natively or from JavaScript) require this repository as a contract to interface with the plugin. Once an interface is published, it should be supported from any release onwards to be backward compatible. Proxies and Stubs, required for over process boundary communication in COMRPC communication are created from the Interface files found in this repository. JSONRPC interface can be automatically created from the COMRPC interface, if indicated in the interface file.

• ThunderNanoServices/ThunderNanoServicesRDK

Dependencies: ***Thunder***, ***ThunderInterfaces***.

This repository holds plugins that realize an interface (COMRPC or JSONRPC) found in the ThunderInterfaces repository. ThunderNanoServices holds plugins that are not being utilized by the RDK/Comcast but only by Metrological customers. ThunderNanoServicesRDK, are plugins developed originally by Metrological but which are now part of the RDK/Comcast stack. The plugins found here are the Next versions to be delivered in RDKServices repository of http://github.com/RDKCentral/rdkservices

• ThunderClientLibraries

Dependencies: ***Thunder***, ***ThunderInterfaces***.

ThunderPlugins cover a great variety of functional areas. There is for example a plugin that monitors the HDMI connections status and the level of content protection on it (HDCP). The plugin is responsible for converting a specific hardware platform to a generic JSONRPC/COMRPC interface capable of running over process boundaries. The consumer of this information is typically a native application, e.g. Amazon, Netflix, Cobalt. These are native apps and thus require a native integration. To make this integration as simple as possible, ThunderClientLibraries offer libraries with a thin C/C++ interface. This simple library can be included in the Amazon, Netflix or Cobalt build and through the implemented thin C/C++ interface hide all the logic involved in JSONRPC/COMRPC communication over process boundaries. This way the integration developer of a native app requires no Thunder knowledge, just the simple C/C++ interface exposed.

There are convenience/client libraries for graphics surfaces (compositorclient), drm (ocdm), hdmi connectivity (displayinfo), device metadata (deviceinfo), playback metadata (playerinfo), security handling (cryptography).

3.Noteworthy Changes in R3

- ## COMRPC Stability

There have been quite some fixes added that improve the COMRPC stability. Mostly in the non happy day scenarios.

- ## Message engine

A new message engine was added that replaces the (duplicate) Tracing, Warning Reporting and SysLog engines. This so all this information is accessible in a consistent way and to increase throughput and performance while on the other side reduce the codesize.
Note: in R3 the old engines are still available and selected by default (build time). In R4 these will be removed completely and replaced by the new message engine.

- ## Debug/Tracing improvements

In numerous places more information is available to help in investigating issues. For example the backtraces are rewritten to be more correct and are now also easier accessible and for example the process name was added to the Job logging.

- ## JSON enhancements

Numerous fixes were made to the JSON parser. For example float parsing was added and the parsing is now fully UTF-8 compliant. (some fixes were cherrypicked to later R2 versions already)

- ## WarningReporting extension

More features were added to WarningReporting. E.g. there are now warnings for jobs that take too long or are stuck waiting to be handled for too long.

- ## WorkerPool/Threadpool improvements

Improvements were made to the Worker- and Threadpool to improve the stability, certainly in non happy day scenarios.

- **Proxy(Pool) improvements**

The Proxy code as well as the ProxyPool (and List, Map etc.) have seen major fixes which improves stability (mostly in the non happy day scenarios) and performance.

- **Improved Frame handling**

Frame coding now handles both little and big endianness correctly and the Frame size can also be specified. This improves usage in different areas like low level network handling and Bluetooth exchanges.

- **Config generation future proof**

The generation of the (Plugin) config files was rewritten. This to make it easier to write and maintain these and also because the old method was not compatible with newer Buildroot versions

- **User/Group options**

An abstraction was added to allow User/Group to be specified for a file/folder handle and an implementation for Linux is provided

- **Hellgrind fixes**

Some changes were made after potential issues were found (low risk, non happy day scenarios) when the hellgrind analysis tool was used.

- **Cyclic buffer improvements**

The cyclicbuffer code was partially rewritten after some potential issues were found.

- **File/Folder path additional security checking**

For security reasons changes were made to the File/Folder path parsing code when used to parse relative paths. This to eliminate the access to files/folder in locations that should be considered inaccessible.

- **Core time improvements**

The Core time handling has been improved to fix some issues with (timezone) handling, make behaviour consistent for multiple Operating Systems and to add Julian data support.

- **Plugin Termination**

Plugin termination was implemented. It is now possible to have a Plugin deactivate automatically when a certain condition (subsystem being unset or set).

- **Versioning support improved**

Thunder now as improved support for versioning. A Major Minor and Patch level were introduced for both the Plugin and JSON RPC interface version. The Plugin version can now be specified in a Plugin metadata section. With this new MetaData section it is now also possible to indicate the precondition and Termination conditions for the plugin and the condition (subsystem) the plugin provides. Cherrypicked to later R2 versions already.

- **Library search path**

It is possible to configure for the a plugin to extend the library search path for that particular plugin. (so for Linux based systems this means to extend LD_LIBRARY_PATH for a plugin). Cherrypicked to later R2 versions already.

- **Unavailable state added**

A plugin can now also have a state “Unavailable”. In the Unavailable state the plugin cannot be activated and the libraries for the plugin can be replaced. Cherrypicked to later R2 versions already.

- **Performance Metrics plugin**

A plugin was added (In ThunderNanoServicesRDK) that makes it possible to out certain performance metrics collected when the plugin is running.

- **Container support enhacements**

Process container support was cleaned up and improved.

- **SocketPort Buffer management**

Buffer management for the socket ports was improved. Kernel buffer sizes can now be adapted per socket.

- ## 64 bits compatibility

64 Bits compatibility improved. 64 Bit Raspberry Pi stack has been tested.

- ## Type traits enhancements

The possibility to use Code constructions that are only executed when they are available but are ignored otherwise (so the code will still compile) has been improved. (in C++ terms this would be called SFINAE)

- ## Smartlink introduction

A Smartlink was introduced to make it really easy to write Client code that can use the COM RPC mechanism to communicate with the Server side. It will also handle the non happy day scenarios. Cherrypicked to later R2 versions already.

- ## Resource monitor improvements

Improvement have been made to the code that measure resources for a component (e.g. memory) and measure this more consistent and precise.

- ## Hide non externals

By default now we hide all the symbols in the export table that are internal. This leads to (much) smaller binaries.

- ## Netlink support

Core now also have support for communicating over Netlink.

- ## Bluetooth

Improvements have been made to the Bluetooth library (L2CAP sockets and HCI socket handling)

- **Controller interface**

The controller interface as extended and now also has a COM RPC interface.

- **JSON RPC Compatibility**

The JSON RPC generation/parsing code has been improved to be more following the JSOPN RPC specification (e.g. single parameter handling)

- **Latitude/Longtitude**

The location subsystem now also supports Latitude/Longtitude

- **Examples extended**

The example code (e.g. for JSON RPC, OOP plugins, COM RPC) have been extended.

- **Untangle Proxy/Stub**

Proxy/Stub code has been separated from the implementation of the service. Proxy/Stubs can now be build without needing to reference the implementation of the service.

- **OCDM improvements**

CBC and CTR decryption support added.

- **CMake improvements**
- **Unit tests extended**
- **Many more general bugfixes**

4.R3 Breaking compatibility

- **Plugin state notification**

The plugin state notification interface was changed and when used in Plugins the plugin will need adaptation (it will not compile against Thunder R3 so will be detected in the build). The change was necessary as with the old interface raceconditions could appear that could not be fixed without changing the interface.

5.Advised to change

When moving to Thunder R3 it is advised to change the service registration for a plugin from the SERVICE_REGISTRATION macro to the new Metadata structure.

6.Current State of R3

The sources can be found at https://github.com/RDKCentral in the repositories:

Thunder	
ThunderInterfaces	
ThunderNanoServices	
ThunderNanoServicesRDK	
ThunderClientLibraries	

7.System identifiers

As Release 3 is not yet frozen the system identifiers for the latest R3 release cannot be specified yet. At the moment of writing the latest R3 version number is 3.4.0. One more minor release is planned that will add new functionality, the OCDM CBC and CTR decryption will be added, so the expected last functional R3 release is 3.5. This document will be updated after the last functional release of R3.

Unique identifier:

Human readable version number:

The unique identifier can be found using a regular browser and surfing to:

http:<IP address of the box>:8080/Service/DeviceInfo

8.Open Issues

None known at the moment.