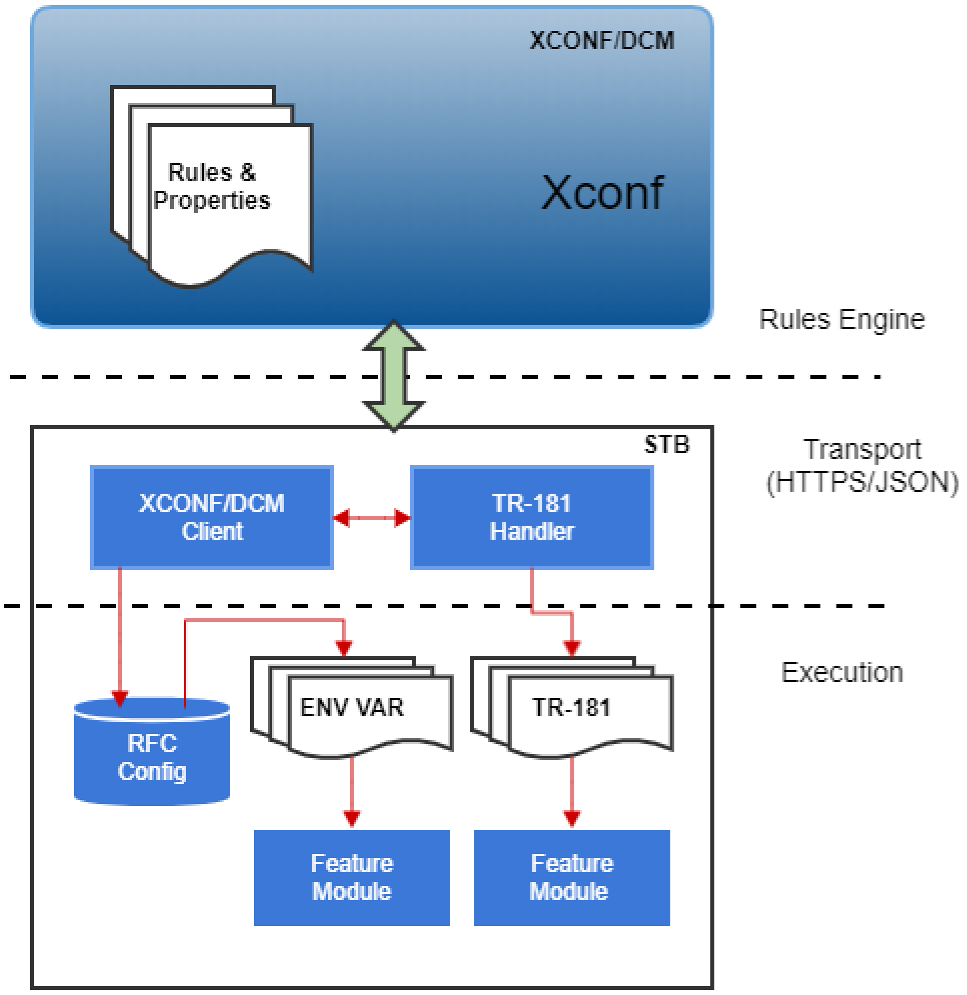
RDK Feature Control - RFC

Overview

Before RFC, the only way to disable a new feature in the field was to rollback to the older firmware. This led to a lot of operational overhead. Also there were lack of options to do a feature deployment in a subset of devices (for eg: only in 100 devices). Also there was lack of options to deliver dynamic configurations to the box Eg: Whitelist of SSH /SNMP servers, Details of downloadable modules..etc

Using RFC

- Enables quicker roll out of features
- Enables a secure channel for delivering run-time configurations to the device
- Ability to control when the feature needs to be enabled/disabled ? Disable now/ Disable during reboot



This approach breaks down the RFC control flow into segments that have specific responsibilities, in accordance with their component design.

- Rules Engine - a passive platform that responds to a specific request, running a defined set of rules against the incoming set of parameters, providing generic data results based on the application of the rules.
- Transport - provides the conduct of signalling and data parameters from client to cloud and back
- Execution - application of the RFC parameters on the client

An RFC transaction begins with the initiation event that causes the rules engine to evaluate a given rule set. In conjunction with the existing telemetry initialization, which is a HTTP GET sent directly from the client to DCM, the RFC request will be sent at a specific point in the client startup process. The response from XCONF/DCM will include the new JSON data structure that defines the feature control data for that specific client.

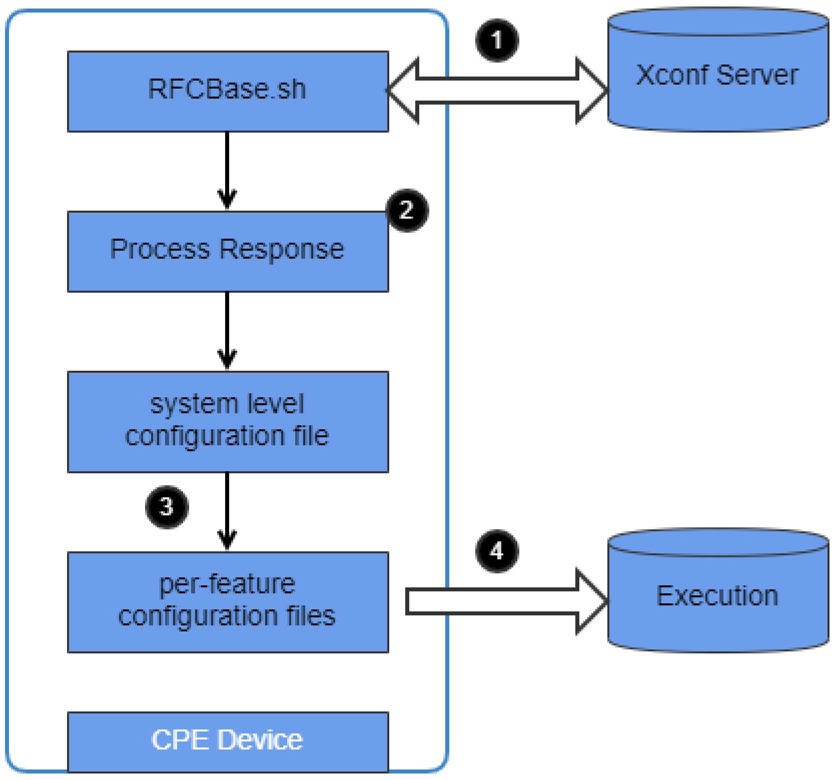
Key aspects of the client RFC processing:

- HTTP Get to XCONF/DCM requesting RFC settings
  Response from DCM is parsed and the configuration data extracted
    - includes "featureControl" data block
    - looks for "effectiveImmediate" flag, evaluates if change in corresponding enable state, triggers reboot
- The JSON config data supports 2 modes for the client control: Environment Variable, or TR-181 data paamaters
- At feature startup, the feature modules will look at the Environment Variable (e.g. RFC_ENABLE_LSA = true) or TR-181 parameter, to determine enable/disable state

Use cases

The primary use case for RFC is the enable/disable of a specific settop feature (e.g. XB Smart Cloud, or LSA on settops), providing a remote control capability to support a progressive roll-out of a feature, as well as roll-back of an already enabled feature. The execution layer supporting the enable /disable can take one of two forms: using a system agent to run a specific process in a linux/Yocto model directly (using systemd or the like), or to signal an existing process to control the behavior of feature specific processing. The Smart Cloud feature in RDKB is a stand-alone process, and the client-side processing of the enable flag in TR-181 will use systemd to start that process. Alternatively, the LSA feature is an extension of the RMF component, and therefore, doesn't fit the stand-alone process model, and so will use a feature-specific method for executing the enable function.

How it works


RFCBase.sh
Process Response
system level configuration file
per-feature configuration files
CPE Device
Xconf Server
Execution
1
2
3
4

1. The RFC process is controlled through rfc-config service
2. RFCBase.sh communicates with xconf server and fetches the predefined feature data.
3. After parsing the feature control JSON messages, RFCBase.sh will create system level & individual configuration files with the key and values.
4. RDK modules pick the environment variables using RFC API or using the script and toggle the relevant feature.