Telemetry 2.0 support in RDKB RPI

Introduction

As part of Telemetry 2.0 , a marker event system has been added to reduce grepping of log files and to improve telemetry handling performance.
Following are major aspects of the marker event system:

Configuration
The XConf/DCM response TelemetryProfile definition has been extended to indicate the source of a marker for report generation is the event system.
When the TelemetryProfile "type" field equals "<event>" the T2 component will not include the marker in the log file grep process. If the "type" field is set to "<event>", then the "content" field is set to the name of the component.
When the T2 component converts a TelemetryProfile to a report profile at .BulkData.Profile.{i}. it creates a Parameter.{i}. entry for each marker . The Use field is set for marker event handling by the T2 component.
Example
Existing marker with log file as the source:

```
"telemetryProfile": [{
    "header": "bootuptime_ClientConnectComplete_split",
    "content": "Client_Connect_complete:",
    "type": "LM.txt.log",
    "pollingFrequency": "0" },
```

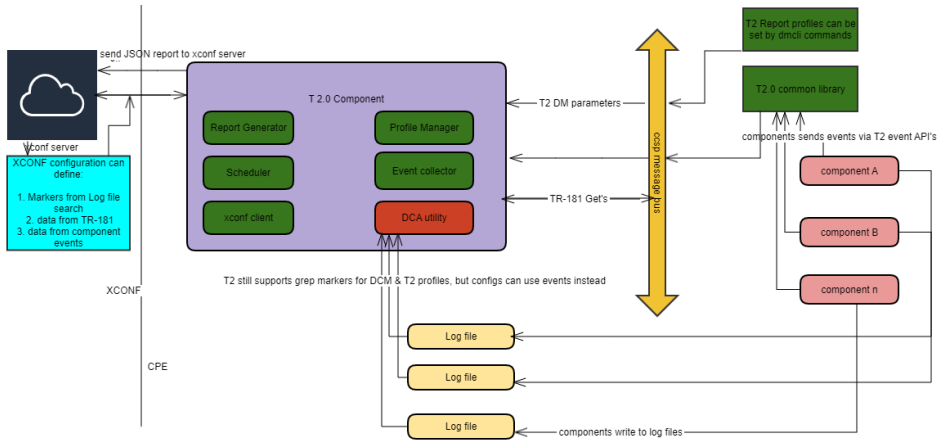
Updated marker with event feed as the source; "type" and "content" field are changed:

```
"telemetryProfile": [{
  "header": "bootuptime_ClientConnectComplete_split",
   "content": "ccsp-lm-lite",
   "type": "<event>",
   "pollingFrequency": "0"
},
```

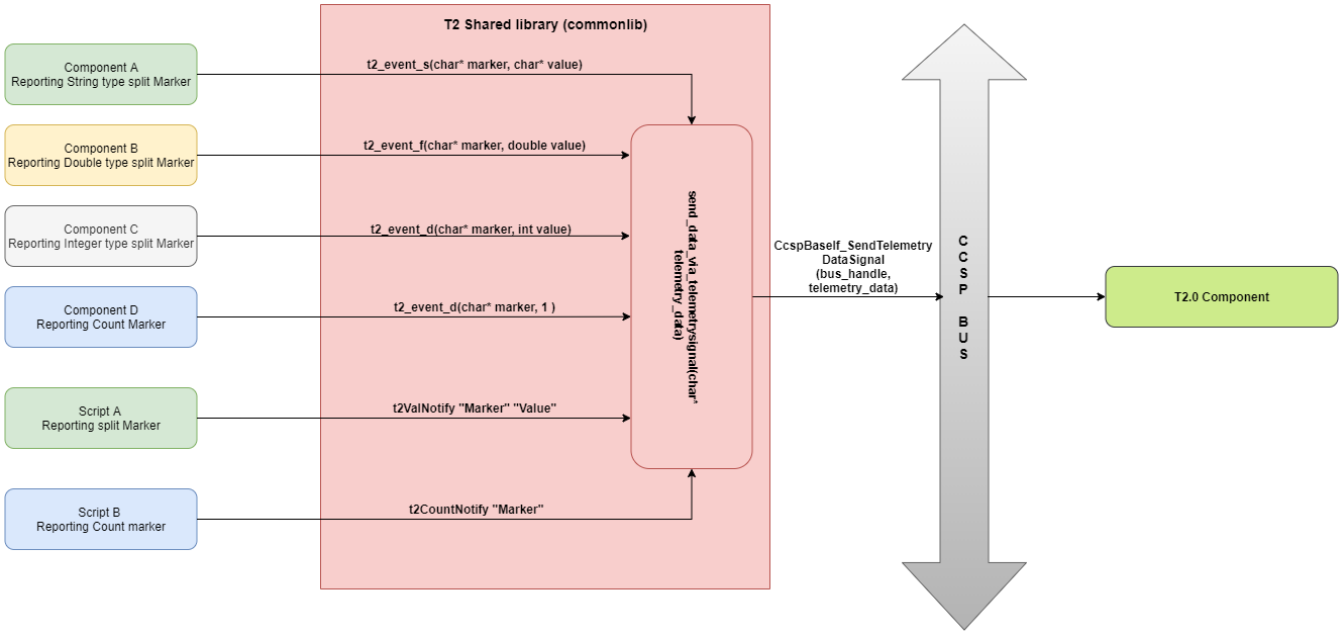
A shared library is used by components to send targeted marker occurrences through an event feed to the T2 component.

Architecture Overview

Telemetry 2.0



Overview of Instrumenting RDKB components with T2 shared library (commonlib) APIs:



T2.0 Common Library

•APIs for component metrics and events

t2_event_s(char*eventName, char* value)
t2_event_d(char*eventName, int value)
t2_event_f(char*eventName, double value)

TR-181 DataModel

S.NO	DataModel	Description	DataType
1	Device.DeviceInfo.X_RDKCENTRAL-COM_RFC.Feature.Telemetry.Enable	Enable the T2 Process	Boolean

2	Device.DeviceInfo.X_RDKCENTRAL-COM_RFC.Feature.Telemetry.Version	'2.0' : Operates only with legacy support '2.0.1' : Operates with T2 report profile and legacy support	String
3	Device.DeviceInfo.X_RDKCENTRAL-COM_RFC.Feature.Telemetry.ConfigURL	https://xconf.xcal.tv/loguploader/getT2Settings	String

T2 Report Profile

S.NO	DataModel	Description	DataType
1	Device.X_RDKCENTRAL-COM_T2.ReportProfiles	Value must be a JSON configuration blob	String
2	Device.X_RDKCENTRAL-COM_T2.ReportProfilesMsgPack	Value must be a JSON configuration blob in base 64 encoded msgpack format	String

Types of Markers

The markers are of 3 types .
1.Split based markers.
2. Count based markers
3. TR-181 based markers.

Marker Type	Sample configuration from xconf	Description with respect to sample configuration
Count based markers	`{"header":"RF_ERROR_IPV4PingFailed","content":"Ping to IPv4 Gateway Address are failed","type":"SelfHeal.txt.0","pollingFrequency":"0"}`	Expects the occurrence count of content "Ping to IPv4 Gateway Address are failed"
Split based markers	`{"header":"bootuptime_ClientConnectComplete_split","content":" Client_Connect_complete:","type":"LM.txt.log","pollingFrequency":"0"}`	Expects the value after content "Client_Connect_complete:"
TR-181 based markers	{"header" : "CMMAC_split","content" : "Device.DeviceInfo.X_COMCAST-COM_CM_MAC"," type" : "<message_bus>","pollingFrequency":"48"}	Markers whose type is configured as "<message_bus>"

In T2.0, the aim is to instrument possible number of split and count based markers from component side. These are termed as event markers. Can be classified as one more type under the classification of markers.
Once a marker is instrumented from component side, its configuration on xconf will be changed from the configured file name to "<event>" in 'type:' section.

Example:

```
{"header":"bootuptime_ClientConnectComplete_split","content":"ccsp-lm-lite","type":<event>","pollingFrequency":"
0"}
```

Find the correct place to report a marker .

Previously in DCA telemetry, a marker is reported based on the xconf configured "content" string - when the content string is found in corresponding configured filename configured under 'type:' section .
/* Refer : {"header": "WIFI_INFO_Hotspot_client_connected", "content": "Added case, Client with:", "type": "Hotspotlog.txt.0","pollingFrequency":"0"} */

- So, find the right place where the content string is being written to the corresponding log file in order to event a marker in T2.0.

In https://code.rdkcentral.com/r/c/rdkb/components/opensource/ccsp/hotspot/+/35833/3/source/hotspotfd/cosa_hotspot_dml.c (Line number 85 gives the idea)

CcspTraceInfo(("Added case, Client with MAC:%s will be added\n", l_cMacAddr));

- Once the place is decided, use the right API to report Marker and values.
  - For markers without "_split" suffix, the marker data is just a count of the number of times the marker is received. In this case, the *t2_event_d* API can be used because the marker data passed to the API is not important.

Example: *t2_event_d("WIFI_INFO_Hotspot_client_connected", 1);* in https://code.rdkcentral.com/r/c/rdkb/components/opensource/ccsp/hotspot/+/35833/3/source/hotspotfd/cosa_hotspot_dml.c

- For markers with "_split" suffix, the marker data is important, so use the API most appropriate to the marker data. For instance, if the marker data is a string, use *t2_event_s*. But if marker data is numeric, use one of *t2_event_d* or *t2_event_f*. Also note that testing must ensure the string used for marker data matches the string expected by legacy telemetry.

Example : *t2_event_s("acs_split", pStr);* in https://code.rdkcentral.com/r/c/rdkb/components/opensource/ccsp/CcspTr069Pa/+/35825/4/source-embedded/DslhManagementServer/ccsp_management_server_pa_api.c line 585

Use appropriate APIs to event markers and values.

In RDKB we have logs coming from both scripts and component's code (code in C). From which markers are reported/grepped.

List of APIs :

- To report markers from components
  - **t2_event_s(char* marker, char* value)** - To send _split marker with string value to T2

Usage: t2_event_s("xh_mac_3_split", "xh_MAC_value");
t2_event_s("xh_mac_3_split", strBuff); /* where strBuff contains the string value to be reported for this marker */
Sample Reviews:

https://code.rdkcentral.com/r/c/rdkb/components/opensource/ccsp/CcspWifiAgent/+/30422/1/source/TR-181/sbapi/wifi_monitor.c where telemetryBuff is a an array of characters declared, reset and copied with string value to be reported.
https://code.rdkcentral.com/r/c/rdkb/components/opensource/ccsp/CcspTr069Pa/+/35825/4/source-embedded/DslhManagementServer/ccsp_management_server_pa_api.c where pStr is a buffer already used in the code.

NOTE: The instrumented component could use a static buffer or do a buffer malloc itself; but T2 common lib makes its own copy regardless, so instrumented component must clean up after itself.

- **t2_event_f(char* marker, double value)** - To send marker with double value to T2
  Usage: t2_event_d("HWREV_split", 2.2);

- **t2_event_d(char* marker, int value)** - To send marker with integer value to T2 (or) to report count based markers
  Usage: t2_event_d("WIFI_INFO_Zero_5G_Clients", 1); /* To report counter based markers-- The value is reported as "1" */
  t2_event_d("Total_5G_clients_split", num_devs); /* To report integer type split markers */
  Sample Review: https://code.rdkcentral.com/r/c/rdkb/components/opensource/ccsp/CcspWifiAgent/+/30422/1/source/TR-181/sbapi/wifi_monitor.c

- To report markers from Scripts.
  - **t2ValNotify "Marker" "Value"** - To report split based markers
    Usage: t2ValNotify "LOAD_AVG_ATOM_split" "$LOAD_AVG_15"
    Sample Review: https://code.rdkcentral.com/r/c/rdkb/components/opensource/ccsp/TestAndDiagnostic/+/30495/1/scripts/log_mem_cpu_info.sh

  - **t2CountNotify "Marker"** - To report count based markers.
    Usage: t2CountNotify "SYS_ERROR_NotRegisteredOnCMTS"
    Sample Review: https://code.rdkcentral.com/r/c/rdkb/components/opensource/ccsp/TestAndDiagnostic/+/30495/1/scripts/corrective_action.sh

Must Check Notes

1.While instrumenting components

- If you are defining a character array buffer to store the value corresponding to marker , ***Make sure Maximum buffer size is allocated, And is reset with '\0' before and after its use.***
  Example: "telemetryBuff" usage in https://code.rdkcentral.com/r/c/rdkb/components/opensource/ccsp/CcspWifiAgent/+/30422/1/source/TR-181/sbapi/wifi_monitor.c
  In many cases, an existing buffer already being built/used within the component can be used rather than necessitating creation of a new buffer; see https://code.rdkcentral.com/r/c/rdkb/components/opensource/ccsp/CcspTr069Pa/+/35825/4/source-embedded/DslhManagementServer/ccsp_management_server_pa_api.c
- To Report count based markers , The value should be mentioned as "1" while using t2_event_d() API.

2. While instrumenting Scripts

Source the utility script /lib/rdk/t2Shared_api.sh

Invoke :

t2ValNotify "Marker" "Value" - To report split based markers

t2CountNotify "Marker" - To report count based markers.

Example: Refer https://code.rdkcentral.com/r/c/rdkb/components/opensource/ccsp/sysint/+/38359/8/uploadRDKBLogs.sh

T2 Report Profiles

A Telemetry 2.0 Report Profile is a configuration, authored in JSON, that can be sent to any RDK device which supports Telemetry 2.0. A Report Profile contains properties that are interpreted by the CPE in order to generate and upload a telemetry report. These properties define the details of a generated report, including:

- Scheduling (how often the report should be generated)

- Parameters (what key/value pairs should be in the report)
- Encoding (the format of the generated report)
- Protocol (protocol to use to send generated report)

Profiles Set Properties

Property	Type	Required
profiles	`array`	Required

profiles

An array of profile objects that each defines a Telemetry 2.0 Report Profile.

`profiles`

- is required
- Type: an array of profile

profiles Constraints

maximum number of items: the maximum number of items for this array is: `10`

Profile

`profile`

- is optional
- Type: `object`

Profile Properties

Property	Type	Required
name	`string`	Required
versionHash	`string`	Required
value	`object`	Required

name

Name of the Profile. This is value is accessible from within the Report Profile as dataModel parameter "Profile.Name".

`name`

- is optional
- Type: `string`

versionHash

Unique value that is expected to change when anything within the Report Profile is changed.

`versionHash`

- is optional
- Type: `string`

value

The JSON representing this Report Profile.

`value`

- is optional
- Type: `object` (JSON object which is a T2 Report Profile)

Property	Type	Required
Description	`string`	Optional
Version	`string`	Optional

Protocol	`string`	Required
EncodingType	`string`	Required
ReportingInterval	`integer`	Optional
ActivationTimeOut	`integer`	Optional
TimeReference	`string`	Optional
GenerateNow	`boolean`	Optional
Parameter	`array`	Required
HTTP	`object`	Optional
JSONEncoding	`object`	Optional

Description

Text describing the purpose of this Report Profile.

`Description`

- is optional
- Type: `string`

Version

Version of the profile. This value is opaque to the Telemetry 2 component, but can be used by server processing to indicate specifics about data available in the generated report.

`Version`

- is optional
- Type: `string`

Protocol

The protocol to be used for the upload of report generated by this profile.

`Protocol`

- is required
- Type: `string`

Protocol Constraints

enum: the value of this property must be equal to one of the following values:

Value	Explanation
`"HTTP"`	When Protocol is equal to HTTP, an HTTP element is expected to occur within the Profile.

EncodingType

The encoding type to be used in the report generated by this profile.

`EncodingType`

- is required
- Type: `string`

EncodingType Constraints

enum: the value of this property must be equal to one of the following values:

Value	Explanation
`"JSON"`	When EncodingType is equal to JSON, a JSONEncoding element is expected to occur within the Profile.

ReportingInterval

The interval, in seconds, at which this profile shall cause a report to be generated.

`ReportingInterval`

- is optional
- Type: `integer`

ActivationTimeOut

The amount of time, in seconds, that this profile shall remain active on the device. This is the amount of time from which the profile is received until the CPE will consider the profile to be disabled. After this time, no further reports will be generated for this report.

`ActivationTimeOut`

- is optional
- Type: `integer`

TimeReference

TBD. Must be value of "0001-01-01T00:00:00Z" for Telemetry 2.0.

`TimeReference`

- is optional
- Type: `string`

TimeReference Default Value

The default value is:

`"0001-01-01T00:00:00Z"`

GenerateNow

When true, indicates that the report for this Report Profile should be generated immediately upon receipt of the profile.

`GenerateNow`

- is optional
- Type: `boolean`

GenerateNow Default Value

The default value is:

`false`

Parameter

An array of objects which defines the data to be included in the generated report. Each object defines the type of data, the source of the data and an optional name to be used as the name (marker) for this data in the generated report.

`Parameter`

- is required
- Type: `object[]`

Parameter Type

`object` (Parameter Definition)

Parameter Constraints

maximum number of items: the maximum number of items for this array is: `800`

HTTP

HTTP Protocol details that will be used when Protocol="HTTP".

`HTTP`

- is optional
- Type: `object`

HTTP Type

`object`

JSONEncoding

JSON Encoding details that will be used when EncodingType="JSON".

`JSONEncoding`

- is optional
- Type: `object`

Examples

Example 1

Simple example to illustrate the JSON structure. The actual "value" object would be in the form of valid JSON representing a T2 report profile.

```
{
    "profiles": [
        {
            "name": "Example_profile_1",
            "versionHash": "profile1Hash111",
            "value": {JSON representing a Report Profile}
        },
        {
            "name": "Example_profile_2",
            "versionHash": "profile2Hash111",
            "value": {JSON representing a Report Profile}
        }
    ]
}
```

Example 2

A profile set containing three profiles, "LMLITE_primer_TEST", "RDKB_CCSPWifi_Profile" and "RDKB_SelfHeal_Profile".

Note that an abbreviated set of parameters is used for each profile for illustrative purposes, therefore, these may not represent desired production profiles.

"value" objects would be in the form of valid JSON representing a T2 report profile.

```
{
    "profiles": [{
            "name": "LMLITE_primer_TEST",
            "versionHash": "lmliteHash111",
            "value": {
                "Description": "A report for a few CCSP-LM-LITE details",
                "Version": "0.1",
                "Protocol": "HTTP",
                "EncodingType": "JSON",
                "ReportingInterval": 60,
                "TimeReference": "0001-01-01T00:00:00Z",
                "ActivationTimeOut": 90,
                "Parameter": [{
                    "type": "grep",
                    "marker": "SYS_SH_TADProcess_restart",
                    "search": "Restarting CcspTandDSsp",
                    "logFile": "LM.txt.0",
                    "use": "absolute"
                }, {
                    "type": "event",
                    "name": "LMLite_caught_wifi_disconnect",
                    "eventName": "WIFI_INFO_clientdisconnect",
                    "component": "ccsp_lmlite",
                    "use": "count",
                    "reportEmpty": false
                }, {
                    "type": "dataModel",
                    "name": "UPTIME",
                    "reference": "Device.DeviceInfo.UpTime",
                    "use": "absolute"
                }],
                "HTTP": {
                    "URL": "http://35.161.239.220/xconf/telemetry_upload.php",
                    "Compression": "None",
                    "Method": "POST",
                    "RequestURIParameter": [{
                        "Name": "profileName",
```

```
                "Reference": "Profile.Name"
            }]
        },
        "JSONEncoding": {
            "ReportFormat": "NameValuePair",
            "ReportTimestamp": "None"
        }
    }
},
{
    "name": "RDKB_CCSPWifi_Profile",
    "versionHash": "wifiHash111",
    "value": {
        "Description": "Report to check WiFi Parameters",
        "Version": "1",
        "Protocol": "HTTP",
        "EncodingType": "JSON",
        "ReportingInterval": 180,
        "TimeReference": "0001-01-01T00:00:00Z",
        "Parameter": [{
            "type": "dataModel",
            "reference": "Device.WiFi.Radio.1.Stats.X_COMCAST-COM_NoiseFloor"
        }, {
            "type": "dataModel",
            "reference": "Device.WiFi.Radio.2.Stats.X_COMCAST-COM_NoiseFloor"
        }, {
            "type": "event",
            "eventName": "2GclientMac_split",
            "component": "ccsp_wifiagent",
            "use": "absolute"
        }, {
            "type": "event",
            "eventName": "5GclientMac_split",
            "component": "ccsp_wifiagent",
            "use": "absolute"
        }, {
            "type": "event",
            "name": "wifiradio WIFI_COUNT",
            "eventName": "WIFI_MAC_1_TOTAL_COUNT:0",
            "component": "ccsp_wifiagent",
            "use": "count",
            "reportEmpty": false
        }, {
            "type": "dataModel",
            "name": "UPTIME",
            "reference": "Device.DeviceInfo.UpTime",
            "use": "absolute"
        }],
        "HTTP": {
            "URL": "http://35.161.239.220/xconf/telemetry_upload.php",
            "Compression": "None",
            "Method": "POST",
            "RequestURIParameter": [{
                "Name": "profileName",
                "Reference": "Profile.Name"
            }, {
                "Name": "reportVersion",
                "Reference": "Profile.Version"
            }]
        },
        "JSONEncoding": {
            "ReportFormat": "NameValuePair",
            "ReportTimestamp": "None"
        }
    }
},
{
    "name": "RDKB_SelfHeal_Profile",
    "hash": "selfHealHash2",
    "value": {
        "Description": "Report to check SelfHeal Parameters",
        "Version": "2",
        "Protocol": "HTTP",
        "EncodingType": "JSON",
        "ReportingInterval": 180,
        "TimeReference": "0001-01-01T00:00:00Z",
        "Parameter": [{
```

```
        "type": "dataModel",
        "name": "Profile",
        "reference": "Device.DeviceInfo.X_RDK.RDKProfileName"
    }, {
        "type": "dataModel",
        "name": "Time",
        "reference": "Device.Time.CurrentLocalTime"
    }, {
        "type": "dataModel",
        "name": "mac",
        "reference": "Device.DeviceInfo.X_COMCAST-COM_WAN_MAC"
    }, {
        "type": "dataModel",
        "name": "erouterIpv4",
        "reference": "Device.DeviceInfo.X_COMCAST-COM_WAN_IP"
    }, {
        "type": "dataModel",
        "name": "erouterIpv6",
        "reference": "Device.DeviceInfo.X_COMCAST-COM_WAN_IPv6"
    }, {
        "type": "dataModel",
        "name": "PartnerId",
        "reference": "Device.DeviceInfo.X_RDKCENTRAL-COM_Syndication.PartnerId"
    }, {
        "type": "dataModel",
        "name": "Version",
        "reference": "Device.DeviceInfo.SoftwareVersion"
    }, {
        "type": "dataModel",
        "name": "AccountId",
        "reference": "Device.DeviceInfo.X_RDKCENTRAL-COM_RFC.Feature.AccountInfo.AccountID"
    }, {
        "type": "dataModel",
        "name": "MAC",
        "reference": "Device.DeviceInfo.X_COMCAST-COM_CM_MAC"
    }, {
        "type": "dataModel",
        "reference": "Profile.Name"
    }, {
        "type": "dataModel",
        "reference": "Profile.Version"
    }, {
        "type": "dataModel",
        "reference": "Device.DeviceInfo.UpTime",
        "use": "absolute"
    }, {
        "type": "event",
        "eventName": "SYS_ERROR_AdvSecurity_NotRunning",
        "component": "test_and_diagnostics",
        "use": "absolute"
    }, {
        "type": "event",
        "eventName": "SYS_SH_lighttpdCrash",
        "component": "test_and_diagnostics",
        "use": "count",
        "reportEmpty": false
    }, {
        "type": "dataModel",
        "name": "WAN_SSH_STATUS",
        "reference": "Device.DeviceInfo.X_RDKCENTRAL-COM_Syndication.WANsideSSH.Enable",
        "use": "absolute"
    }],
    "HTTP": {
        "URL": "http://35.161.239.220/xconf/telemetry_upload.php",
        "Compression": "None",
        "Method": "POST",
        "RequestURIParameter": [{
            "Name": "deviceId",
            "Reference": "Device.DeviceInfo.X_COMCAST-COM_CM_MAC"
        }, {
            "Name": "profileName",
            "Reference": "Profile.Name"
        }]
    },
    "JSONEncoding": {
        "ReportFormat": "NameValuePair",
        "ReportTimestamp": "None"
```

```
                }
            }
        }
    ]
}
```

Example 3

Send an empty set of profiles to remove all Telemetry 2.0 profiles from a device.

```
{
    "profiles": []
}
```

Example 4

This is an example of a complete Report Profile. The "Parameter" property defines the data to be gathered and reported.

Notice that there are three types of data supported: dataModel, event, and grep. The other properties define scheduling, protocols and encoding of the generated report.

```
{
    "Description": "T2.0 WiFi data",
    "Version": "0.1",
    "Protocol": "HTTP",
    "EncodingType": "JSON",
    "ReportingInterval": 900,
    "TimeReference": "0001-01-01T00:00:00Z",
    "Parameter": [{
            "type": "dataModel",
            "name": "MAC",
            "reference": "Device.DeviceInfo.X_COMCAST-COM_CM_MAC"
        },
        {
            "type": "grep",
            "marker": "WIFI_BYTESSENTCLIENTS_1",
            "search": "WIFI_BYTESSENTCLIENTS_1:",
            "logFile": "wifihealth.txt"
        },
        {
            "type": "grep",
            "marker": "WIFI_BYTESSENTCLIENTS_2",
            "search": "WIFI_BYTESSENTCLIENTS_2:",
            "logFile": "wifihealth.txt"
        },
        {
            "type": "event",
            "eventName": "WIFI_CWconfig_1_split",
            "component": "ccsp-wifi-agent"
        },
        {
            "type": "event",
            "eventName": "WIFI_CWconfig_2_split",
            "component": "ccsp-wifi-agent"
        },
        {
            "type": "dataModel",
            "reference": "Device.WiFi.Radio.{i}.OperatingChannelBandwidth"
        },
        {
            "type": "dataModel",
            "name": "WIFI_NF_1_split",
            "reference": "Device.WiFi.Radio.1.Stats.X_COMCAST-COM_NoiseFloor"
        },
        {
            "type": "dataModel",
            "name": "WIFI_NF_2_split",
            "reference": "Device.WiFi.Radio.2.Stats.X_COMCAST-COM_NoiseFloor"
        }

    ],
    "HTTP": {
        "URL": "http://35.161.239.220/xconf/telemetry_upload.php",
        "Compression": "None",
        "Method": "POST",
        "RequestURIParameter": [{
                "Name": "profileName",
                "Reference": "Profile.Name"
            },
            {
                "Name": "reportVersion",
                "Reference": "Profile.Version"
            }
        ]

    },
    "JSONEncoding": {
        "ReportFormat": "NameValuePair",
        "ReportTimestamp": "None"
    }

}
```


T2 ReportProfilesMsgPack

we will add msgPack support to T2.0, such that the T2.0 report profiles will be received in msgPack format. The T2.0 component must unpack the msgPacked data and use it to create the internal structures to represent the active report profiles.T2 Report Profiles can still be authored in JSON. They will just need to be sent through a msgPack and base64 process, like at https://toolslick.com/conversion/data/json-to-messagepack#, to get msgPacked data that has been base64 encoded. Choose Output Type "Base 64".

1. Device.X_RDKCENTRAL-COM_T2.ReportProfilesMsgPack as follows:
    a. Parameter Name: Device.X_RDKCENTRAL-COM_T2.ReportProfilesMsgPack
    Type: base64 (base64-encoded msgPack)
    Access: Read and Write access
    Persistence: Not persisted
    Factory default: Null string
    Usage: When set, T2.0 will use the value of this parameter to configure its T2 active report profiles, as defined in Configuring Active T2.0 Profiles
2. The T2.0 Parameter Device.X_RDKCENTRAL-COM_T2.ReportProfiles that accepts T2 report profiles in JSON format will be deprecated. While deprecated, if report profiles are received via both ReportProfiles and ReportProfilesMsgPack, the last configuration received will be respected.
3. The T2.0 Component must continue to concurrently support the legacy telemetry report profile received from XConf DCM response in JSON format.
4. **Configuring Active Profiles**
    a. T2.0 component must decode the base64 encoded Device.X_RDKCENTRAL-COM_T2.ReportProfilesMsgPack value to get the msgPack binary data
    b. T2.0 component must process the msgPack binary data
        i. msgPack formatted report profiles must be processed according to the msgPack specification: https://github.com/msgpack/msgpack/blob/master/spec.md#type-system