TECHNICAL REPORT

TR-157

Component Objects for CWMP

Issue: 1 Amendment 10
Issue Date: November 2015

Notice

The Broadband Forum is a non-profit corporation organized to create guidelines for broadband network system development and deployment. This Broadband Forum Technical Report has been approved by members of the Forum. This Broadband Forum Technical Report is not binding on the Broadband Forum, any of its members, or any developer or service provider. This Broadband Forum Technical Report is subject to change, but only with approval of members of the Forum. This Technical Report is copyrighted by the Broadband Forum, and all rights are reserved. Portions of this Technical Report may be copyrighted by Broadband Forum members.

THIS SPECIFICATION IS BEING OFFERED WITHOUT ANY WARRANTY WHATSOEVER, AND IN PARTICULAR, ANY WARRANTY OF NONINFRINGEMENT IS EXPRESSLY DISCLAIMED. ANY USE OF THIS SPECIFICATION SHALL BE MADE ENTIRELY AT THE IMPLEMENTER'S OWN RISK, AND NEITHER THE FORUM, NOR ANY OF ITS MEMBERS OR SUBMITTERS, SHALL HAVE ANY LIABILITY WHATSOEVER TO ANY IMPLEMENTER OR THIRD PARTY FOR ANY DAMAGES OF ANY NATURE WHATSOEVER, DIRECTLY OR INDIRECTLY, ARISING FROM THE USE OF THIS SPECIFICATION.

Broadband Forum Technical Reports may be copied, downloaded, stored on a server or otherwise re-distributed in their entirety only, and may not be modified without the advance written permission of the Broadband Forum.

The text of this notice must be included in all copies of this Broadband Forum Technical Report.

Issue History

Issue Number	Approval Date	Publication Date	Issue Editor	Changes
Issue 1	March 2009		John Blackford, 2Wire, Inc. TimSpets, Westell	Original
Issue 1 Amendment 1	September 2009		John Blackford, 2Wire, Inc.	Addition of SupportedDataModel component.
Issue 1 Amendment 2	May 2010		John Blackford, 2Wire, Inc.	Support for TR-181 Issue 2.
Issue 1 Amendment 3	November 2010		John Blackford, Pace Heather Kirksey, Alcatel-Lucent	Support for Software Module Management
Issue 1 Amendment 4	July 2011			Changes to XML only
Issue 1 Amendment 5	November 2011		Tim Carey, Alcatel-Lucent Heather Kirksey, Alcatel-Lucent	Support for Location and Fault Management
Issue 1 Amendment 10	9 November 2015	13 November 2015	Tim Carey, Alcatel-Lucent	Support for HTTP based Bulk Data Collection

Comments or questions about this Broadband Forum Technical Report should be directed to help@broadband-forum.org.

Editor	Tim Carey	Alcatel-Lucent
Broadband User Services Work Area Directors	John Blackford Jason Walls	Pace QA Cafe

Table of Contents

List of Tables

List of Figures

Executive Summary

The architecture of TR-069 [1] and TR-106 [2] enables device management of CPE devices in the customer's home, including the home gateway, and devices behind it.

This Technical Report defines additional management objects for use in CWMP managed devices. The objects can exist at the top level of a hierarchy, or in some cases within an existing object. The objects are intended for use in all CWMP root objects (both Device and InternetGatewayDevice). The objects define varying functionality, diagnostics, etc., that are agnostic to the type of device.

The additional management objects defined in this Technical Report includes the following:

Enhanced device diagnostic and monitoring capabilities - These enhanced features include the ability to monitor device memory and process status as well as reporting of temperature sensor status and alarms. Two diagnostic tests have also been added: Namespace Lookup and hardware-specific self-test.

Autonomous Transfer and Multi-cast Download Policy Configuration - This specification completes the additions to CWMP undertaken in collaboration with DVB to ensure TR-069's ability to meet the needs of IP video environments. In TR-069 [1] capabilities for multi-cast download and autonomous transfers were added to the CWMP protocol; in this Technical Report, objects have been added for managing the policies around autonomous transfer reporting and configuring the multicast download availability.

Simple Firewall - Simple firewall management has been defined in this specification.

USB Hosts - This specification contains objects that enable the remote management of USB Hosts and policies for the behavior of attached USB devices.

UPnP and DLNA discovery - UPnP is a widely deployed home networking technology; DLNA digital home servers and digital home players use UPnP technology to provide content streaming and sharing across devices in the home. Objects defined in this specification enable the reporting of UPnP and DLNA devices and capabilities in the home network in order to give service providers increased visibility into the subscriber home.

Periodic Stats - The periodic stats object allows for the collection and reporting of performance monitoring data for TR-069 enabled devices.

Supported Data Model – This table lists all of the Device Type (as defined in TR-106 [2]) instances that make up the device's entire supported data model and thus allows an ACS to easily discover the device's supported data model.

Software Module Management – These objects enable the management of software modules on a device in order to allow service providers to deploy dynamic applications and services. The capabilities include configuring and managing Execution Environments, Deployment Units, and Execution Units.

Location Management – These objects enable the management of location data within a device.

Fault Management – These objects enable allows for the logging and reporting of alarms and events within a device.

HTTP Bulk Data Collection – This annex describes how to use the Bulk Data Collection objects for transport over HTTP.

1 Purpose and Scope

1.1 Purpose

The purpose of TR-157 is to provide Component Objects for CWMP.

A Component Object is defined as an object and their contained parameters intended for use in any applicable CWMP root data model (both Device). The object(s) can reside at the top level or an appropriate sub-object level.

1.2 Scope

TR-157 defines Component Objects for use in CWMP managed devices for all root data models. The current root data models are Device:1 defined in TR-181 Issue 1 [3], and Device:2 defined in TR-181 Issue 2 [4].

Sections containing "Theory of Operations" for Component Objects are located in the appendices.

2 References and Terminology

2.1 Conventions

In this Technical Report several words are used to signify the requirements of the specification. These words are often capitalized.

MUST	This word, or the term "REQUIRED", means that the definition is an absolute requirement of the specification.
MUST NOT	This phrase means that the definition is an absolute prohibition of the specification.
SHOULD	This word, or the adjective "RECOMMENDED", means that there could exist valid reasons in particular circumstances to ignore this item, but the full implications need to be understood and carefully weighed before choosing a different course.
SHOULD NOT	This phrase, or the phrase "NOT RECOMMENDED" means that there could exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications need to be understood and the case carefully weighed before implementing any behavior described with this label.
MAY	This word, or the adjective "OPTIONAL", means that this item is one of an allowed set of alternatives. An implementation that does not include this option MUST be prepared to inter-operate with another implementation that does include the option.

2.2 References

The following references constitute provisions of this Technical Report. At the time of publication, the editions indicated were valid. All references are subject to revision; users of this Technical Report are therefore encouraged to investigate the possibility of applying the most recent edition of the references listed below. A list of currently valid Broadband Forum Technical Reports is published at www.broadband-forum.org.

Document	Title	Source	Year
[1] TR-069 Amendment 5	*CPE WAN Management Protocol*	Broadband Forum	2013
[2] TR-106 Amendment 7	*Data Model Template for TR-069-Enabled Devices*	Broadband Forum	2013
[3] TR-181 Issue 1	*Device Data Model for TR-069*	Broadband Forum	2010
[4] TR-181 Issue 2 Amendment 10	*Device Data Model for TR-069*	Broadband Forum	2015
[5] TR-098 Amendment 1	*Internet Gateway Device Data Model for TR-069* *(DEPRECATED)*	Broadband Forum	2014
[6] TR-104 Issue 2	*Provisioning Parameters for VoIP CPE*	Broadband Forum	2014
[7] RFC 4122	*A Universally Unique IDentifier (UUID) URN Namespace*	IETF	2005
[8] RFC 5491	*GEOPRIV Presence Information Data Format Location Object (PIDF-LO) Usage Clarification, Considerations, and Recommendation*	IETF	2009
[9] RFC 5139	*Revised Civic Location Format for Presence Information Data Format Location Object (PIDF-LO)*	IETF	2008
[10] RFC 4119	*A Presence-based GEOPRIV Location Object Format*	IETF	2005
[11] IETF draft	*Relative Location Representation – draft-ietf-*	IETF	

	geopriv-relative-location-00		
[12] IANA Method Tokens	*Method Tokens*	IANA	2008
[13] RFC 4479	*A Data Model for Presence*	IETF	2006
[14] GML 3.2.1	*OpenGIS Geography Markup Language (GML) Encoding Standard*	Open Geospatial Consortium (OGC)	
[15] TR-232	*Bulk Data Collection*	Broadband Forum	2012
[16] RFC 4180	*Common Format and MIME Type for Comma-Separated Values (CSV) Files*	IETF	2005
[17] RFC 7159	*The JavaScript Object Notation (JSON) Data Interchange Format.*	IETF	2014
[18] RFC 2616	*Hypertext Transfer Protocol – HTTP/1.1*	IETF	1999
[19] RFC 6066	*Transport Layer Security (TLS) Extensions: Extension Definitions*	IETF	2011
[20] RFC 6125	*Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)*	IETF	2011
[21] RFC 5246	*The Transport Layer Security (TLS) Protocol, Version 1.2*	IETF	2008

2.3 Definitions

The following terminology is used throughout this Technical Report:

ACS	Auto-Configuration Server. This is a component in the broadband network responsible for auto-configuration of the CPE for advanced services.
Action	An explicitly triggered transition in the software module state model (see Appendix II); e.g. Install, Update, Uninstall, Start, Stop, etc.
CPE	Customer Premises Equipment; refers to any TR-069-enabled device and therefore covers both Internet Gateway devices and LAN-side end devices.

CWMP	CPE WAN Management Protocol. Defined in TR-069 [1], CWMP is a communication protocol between an ACS and CPE that defines a mechanism for secure auto-configuration of a CPE and other CPE management functions in a common framework.
Deployment Unit	An entity that can be individually deployed on the Execution Environment. A Deployment Unit can consist of functional Execution Units and/or configuration files and/or other resources
Execution Environment	A software platform that enables the dynamic loading and unloading of software modules. Some Execution Environments enable the sharing of resources amongst modules. Typical examples include Linux, OSGi, .NET, and Java ME. There will likely be one primary Execution Environment on each device, and other "layered" Execution Environments can also be exposed (e.g. OSGi on top of Linux).
Execution Unit	A functional entity that, once started, initiates processes to perform tasks or provide services, until it is stopped. Execution Units are deployed by Deployment Units. The following list of concepts could be considered an Execution Unit: services, scripts, software components, libraries, etc.
Software Module	The common term for all software (other than firmware) that will be installed on an Execution Environment, including the concepts of Deployment Units and Execution Units.

2.4 Abbreviations

This Technical Report defines the following abbreviations:

CPE	Customer Premise Equipment
CPU	Central Processing Unit
DDD	Device Description Document
DLNA	Digital Living Network Alliance
DNS	Domain Name System
DU	Deployment Unit
EE	Execution Environment
EU	Execution Unit
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol over Secure Socket Layer
IGD	Internet Gateway Device
LAN	Local Area Network
NAT	Network Address Translation
QoS	Quality of Service
RAM	Random Access Memory
SSDP	Simple Service Discovery Protocol
TCP	Transmission Control Protocol
TR	Technical Report
URL	Universal Resource Locator
USB	Universal Serial Bus
USB-IF	USB Implementer's Forum
USN	Unique Service Name
UTC	Coordinated Universal Time
UUID	Universally Unique Identifier
WAN	Wide Area Network
WG	Working Group
XML	Extensible Markup Language

3 Technical Report Impact

3.1 Energy Efficiency

TR-157 has no impact on energy efficiency.

3.2 IPv6

TR-157 has no impact on IPv6 support and compatibility.

3.3 Security

There are no relevant security issues relating to TR-157.

4 CWMP Common Component Parameter Definitions

The normative definitions of the CWMP common component data model, along with links to the associated XML and HTML files, are defined in Table 1.

Because new minor versions of the CWMP component data model can be defined without re-publishing this Technical Report, the table is not necessarily up-to-date. An up-to-date version of the table can always be found at http://www.broadband-forum.org/cwmp.

Table 1 – CWMP Common Component Data Model Versions

DM Instance	XML and HTML
tr-157-1-1.xml	http://broadband-forum.org/cwmp/tr-157-1-1.xml
	http://broadband-forum.org/cwmp/tr-157-1-1.html
tr-157-1-2.xml	http://broadband-forum.org/cwmp/tr-157-1-2.xml
	http://broadband-forum.org/cwmp/tr-157-1-2.html
tr-157-1-3.xml	http://broadband-forum.org/cwmp/tr-157-1-3.xml
	http://broadband-forum.org/cwmp/tr-157-1-3.html
tr-157-1-4.xml	http://broadband-forum.org/cwmp/tr-157-1-4.xml
	http://broadband-forum.org/cwmp/tr-157-1-4.html
tr-157-1-5.xml	http://broadband-forum.org/cwmp/tr-157-1-5.xml
	http://broadband-forum.org/cwmp/tr-157-1-5.html
tr-157-1-6.xml	http://broadband-forum.org/cwmp/tr-157-1-6.xml
	http://broadband-forum.org/cwmp/tr-157-1-6.html
tr-157-1-7.xml	http://broadband-forum.org/cwmp/tr-157-1-7.xml
	http://broadband-forum.org/cwmp/tr-157-1-7.html
tr-157-1-8.xml	http://broadband-forum.org/cwmp/tr-157-1-8.xml
	http://broadband-forum.org/cwmp/tr-157-1-8.html
tr-157-1-9.xml	http://broadband-forum.org/cwmp/tr-157-1-9.xml
	http://broadband-forum.org/cwmp/tr-157-1-9.html
tr-157-1-10.xml	http://broadband-forum.org/cwmp/tr-157-1-10.xml
	http://broadband-forum.org/cwmp/tr-157-1-10.html

Annex A: HTTP Bulk Data Collection

This section discusses the Theory of Operation for the collection and transfer of bulk data using TR-069 [1], HTTP and the BulkData object defined in the Root data model.

A.1 Overview

This section describes a method to collect within the CPE and transfer collected data to a Bulk Data Collector in the service provider network utilizing:

- HTTP/HTTPS for the transfer of collected data
- CSV and JSON for the encoding of collected data to be transferred

The CPE configuration that enables the collection of bulk data using HTTP is defined using the BulkData component objects defined within this specification.

A.2 Enabling HTTP/HTTPS Bulk Data Communication

HTTP/HTTPS communication between the CPE and Bulk Data Collector is enabled by configuring the BulkData.Profile object for the HTTP/HTTPS transport protocol adding a new BulkData.Profile object instance using the AddObject RPC and configuring it with SetParameterValue RPC. For example:

- .BulkData.Profile.1
- .BulkData.Profile.1.Enable=true
- .BulkData.Profile.1.Protocol = "HTTP"
- .BulkData.Profile.1.ReportingInterval = 300
- .BulkData.Profile.1.TimeReference = "0001-01-01T00:00:00Z"
- .BulkData.Profile.1.HTTP.URL = "https://bdc.acme.com/somedirectory"
- .BulkData.Profile.1.HTTP.Username = "username"
- .BulkData.Profile.1.HTTP.Password = "password"
- .BulkData.Profile.1.HTTP.Method = "POST"
- .BulkData.Profile.1.HTTP.UseDateHeader = true

The configuration above defines a profile that transfers data from the CPE to the Bulk Data Collector (bdc.acme.com/somedirectory) using secured HTTP. In addition the CPE will provide authentication credentials (username, password) to the Bulk Data Collector, if requested by the Bulk Data Collector. Finally the CPE establishes a communication session with the Bulk Data Collector every 300 seconds in order to transfer the data defined by the BulkData Report object instance.

Once the communication session is established between the CPE and Bulk Data Collector the data is transferred from the CPE using the POST HTTP method with a HTTP Date header and no compression.

In many scenarios CPEs will utilize "chunked" transfer codings. As such, the Bulk Data Collector MUST support the HTTP transfer-coding value of "chunked".

A.2.1 Use of the URI Query Parameters

The HTTP Bulk Data transfer mechanism allows parameters to be used as HTTP URI query parameters. This is useful when Bulk Data Collector utilizes the specific parameters that the CPE reports for processing (e.g., logging, locating directories) without the need for the Bulk Data Collector to parse the data being transferred.

The CPE MUST transmit the device's Manufacturer OUI, Product Class and Serial Number as part of the URI query parameters. The data model parameters are encoded as:

- .DeviceInfo.ManufacturerOUI -> oui
- .DeviceInfo.ProductClass -> pc
- .DeviceInfo.SerialNumber -> sn

As such the values of the device’s OUI, Serial Number and Product Class are formatted in the HTTP request URI as follows:
POST https://bdc.acme.com/somedirectory?oui=00256D&pc=Z&sn=Y

Configuring the URI query parameters for other parameters requires that instances of a BulkData.Profile.{i}.HTTP.RequestURIParameter object instance be created and configured with the requested parameters. The additional parameters are appended to the required URI query parameters.

Using the example to add the device's current local time to the required URI parameters, the HTTP request URI would be as follows:
POST https://bdc.acme.com/somedirectory?oui=00256D&pc=Z&sn=Y&ct=2015-11-01T11:12:13Z by setting the following parameters using the AddObject and SetParameterValues RPC as follows:

- .BulkData.Profile.1.HTTP.RequestURIParameter 1.Name ="ct"
- .BulkData.Profile.1.HTTP.RequestURIParameter.1.Reference ="Device.Time.CurrentLocalTime"

A.2.2 Use of HTTP Status Codes

The Bulk Data Collector uses standard HTTP status codes, defined in the HTTP specification, to inform the CPE whether a bulk data transfer was successful. The HTTP status code will be set in the response header by the Bulk Data Collector. For example, "200 OK" status code indicates an upload was processed successfully, "202 Accepted" status code indicates that the request has been accepted for processing, but the processing has not been completed, "401 Unauthorized" status code indicates user authentication failed and a "500 Internal Server Error" status code indicates there is an unexpected system error.

A.2.2.1 HTTP Retry Mechanism

When the CPE receives an unsuccessful HTTP status code and the HTTP retry behavior is enabled, the CPE MUST try to redeliver the data. The retry mechanism employed for the transfer of bulk data using HTTP uses the same algorithm as the CWMP session retry defined in Section 3.2.1.1 of TR-069 [1].

When retrying a failed transfer, the CPE MUST keep track of the number of times it has attempted to retry a failed data transfer attempt. A CPE MUST retry a failed transfer after waiting for an interval of time specified in Table 2. The CPE MUST choose the wait interval by randomly selecting a number of seconds from the range given by the data transfer retry count. When retrying a failed data transfer after an intervening reboot, the CPE MUST reset the wait intervals it chooses from as though it were making its first data transfer retry attempt.

The wait interval range is controlled by two Parameters, the minimum wait interval and the interval multiplier, each of which corresponds to a data model Parameter, and which are described in the table below.

Descriptive Name	Symbol[1]	Default	Data Model Parameter Name
Minimum wait interval	m	5 seconds	.BulkData.Profile.{i}.HTTP.RetryMinimumWaitInterval
Interval multiplier	k	2000	.BulkData.Profile.{i}.HTTP.RetryIntervalMultiplier

Beginning with the tenth retry attempt, the CPE MUST choose from the fixed maximum range shown in Table 2. The CPE MUST continue to retry a failed data transfer until it is successfully terminated or until the next reporting interval for the data transfer becomes effective. Section A.3.3 discusses the processing of failed data transfers. Once the data is transferred successfully, the CPE MUST reset the data transfer retry count to zero and no longer apply session retry policy to determine when to initiate the next data transfer.

Table 2 – Data Transfer Retry Wait Intervals

Data Transfer Retry Count	Default Wait Interval Range (min-max seconds)	Actual Wait Interval Range (min-max seconds)
#1	5-10	$m – m.(k/1000)$
#2	10-20	$m.(k/1000) – m.(k/1000)^2$
#3	20-40	$m.(k/1000)^2 – m.(k/1000)^3$
#4	40-80	$m.(k/1000)^3 – m.(k/1000)^4$

[1] These symbols are used in Table 2 – Data Transfer Retry Wait IntervalsTable 2.

Data Transfer Retry Count	Default Wait Interval Range (min-max seconds)	Actual Wait Interval Range (min-max seconds)
#5	80-160	$m.(k/1000)^4$ – $m.(k/1000)^5$
#6	160-320	$m.(k/1000)^5$ – $m.(k/1000)^6$
#7	320-640	$m.(k/1000)^6$ – $m.(k/1000)^7$
#8	640-1280	$m.(k/1000)^7$ – $m.(k/1000)^8$
#9	1280-2560	$m.(k/1000)^8$ – $m.(k/1000)^9$
#10 and subsequent	2560-5120	$m.(k/1000)^9$ – $m.(k/1000)^{10}$

A.2.3 Use of TLS and TCP

The use of TLS to transport the HTTP Bulk Data is RECOMMENDED, although the protocol MAY be used directly over a TCP connection instead. If TLS is not used, some aspects of security are sacrificed. Specifically, TLS provides confidentiality and data integrity, and allows certificate-based authentication in lieu of shared secret-based authentication.
Certain restrictions on the use of TLS and TCP are defined as follows:

- The CPE MUST support TLS 1.2 [21] (or a later version).
- If the Collection Server URL has been specified as an HTTPS URL, the CPE MUST establish secure connections to the Collection Server, and MUST start the TLS session negotiation with TLS 1.2 (or, if supported, a later version).

  Note – If the Collection Server does not support the version with which the CPE establishes the connection, it might be necessary to negotiate an earlier TLS 1.x version, or even SSL 3.0. This implies that the CPE has to support the mandatory cipher suites for all supported TLS or SSL versions.

  Note – TLS_RSA_WITH_AES_128_CBC_SHA is the only mandatory TLS 1.2 cipher suite.

- The CPE SHOULD use the RFC 6066 [19] Server Name TLS extension to send the host portion of the Collection Server URL as the server name during the TLS handshake.
- If TLS 1.2 (or a later version) is used, the CPE MUST authenticate the Collection Server using the certificate provided by the Collection Server. Authentication of the Collection Server requires that the CPE MUST validate the certificate against a root certificate. To validate against a root certificate, the CPE MUST contain one or more trusted root certificates that are either pre-loaded in the CPE or provided to the CPE by a secure means outside the scope of this specification. If as a result of an HTTP redirect, the CPE is attempting to access a Collection Server at a URL different from its pre-configured Collection Server URL, the CPE MUST validate the Collection Server certificate using the redirected Collection Server URL rather than the pre-configured Collection Server URL.
  - If the host portion of the Collection Server URL is a DNS name, this MUST be done according to the principles of RFC 6125 [20], using the host portion of the Collection Server URL as the *reference identifier*.

- If the host portion of the Collection Server URL is an IP address, this MUST be done by comparing the IP address against any *presented identifiers* that are IP addresses.

  Note – the terms "reference identifier" and "presented identifier" are defined in RFC 6125.
  Note – wildcard certificates are permitted as described in RFC 6125.

- A CPE capable of obtaining absolute time SHOULD wait until it has accurate absolute time before contacting the Collection Server. If a CPE for any reason is unable to obtain absolute time, it can contact the Collection Server without waiting for accurate absolute time. If a CPE chooses to contact the Collection Server before it has accurate absolute time (or if it does not support absolute time), it MUST ignore those components of the Collection Server certificate that involve absolute time, e.g. not-valid-before and not-valid-after certificate restrictions.
- Support for CPE authentication using client-side certificates is NOT RECOMMENDED. Instead, the Collection Server SHOULD authenticate the CPE using HTTP basic or digest authentication to establish the identity of a specific CPE.

A.3 Encoding of Bulk Data

Bulk Data that is transferred to the Bulk Data Collector from the CPE using HTTP/HTTPS is encoded using a specified encoding type. For HTTP/HTTPS the supported encoding types are CSV and JSON. The encoding type is sent a media type with the report format used for the encoding. For CSV the media type is **text/csv** as specified in RFC 4180 [16] and for JSON the media type is **application/json** as specified in RFC 7159 [17]. For example a CSV encoded report using charset=UTF-8 would have the following Content-Type header:
Content-Type: text/csv; charset=UTF-8

The "media-type" field and "charset" parameters MUST be present in the Content-Type header.

In addition the report format that was used for encoding the report is included as a HTTP custom header with the following format:
BBF-Report-Format: <ReportFormat>
The <ReportFormat> field is represented as a token.

For example a CSV encoded report using a ReportFormat for ParameterPerRow would have the following BBF-Report-Format header:
BBF-Report-Format: "ParameterPerRow"

The BBF-Report-Format custom header MUST be present when transferring data to the Bulk Data Collector from the CPE using HTTP/HTTPS.

A.3.1 Using Wildcards to Reference Object Instances in the Report

When the CPE supports the use of the Wildcard value "*" in place of instance identifiers for the Reference parameter, then all object instances of the referenced parameter are encoded. For

example to encode the "BroadPktSent" parameter for all object instances of the MoCA Interface object the following will be configured:

- .BulkData.Profile.1.Parameter.1.Name = ""
- .BulkData.Profile.1.Parameter.1.Reference = "Device.MoCA.Interface.*.Stats.BroadPktSent"

A.3.2 Using Alternative Names in the Report

Alternative names can be defined for the parameter name in order to shorten the name of the parameter. For example instead of encoding the full parameter name "Device.MoCA.Interface.1.Stats.BroadPktSent" could be encoded with a shorter name "BroadPktSent". This allows the encoded data to be represented using the shorter name. The following depicts how this would be configured:

- .BulkData.Profile.1.Parameter.1.Name = "BroadPktSent"
  .BulkData.Profile.1.Parameter.1.Reference = "Device.MoCA.Interface.1.Stats.BroadPktSent"

In the scenario where there are multiple instances of a parameter (e.g., "Device.MoCA.Interface.1.Stats.BroadPktSent", "Device.MoCA.Interface.2.Stats.BroadPktSent") in a Report, the content of the Name parameter SHOULD be unique (e.g., BroadPktSent1, BroadPktSent2).

A.3.2.1 Using Object Instance Wildcards and Parameter Partial Paths with Alternative Names

Wildcards for Object Instances can be used in conjunction with the use of alternative names by reflecting object hierarchy of the value of the Reference parameter in the value of the Name parameter.

When the value of the Reference parameter uses a wildcard for an instance identifier, the value of the Name parameter (as used in a report) MUST reflect the wild-carded instance identifiers of the parameters being reported on. Specifically, the value of the Name parameter MUST be appended with a period (.) and then the instance identifier. If the value of the Reference parameter uses multiple wildcard then each wild-carded instance identifier MUST be appended in order from left to right.

For example, for a device to report the Bytes Sent for the Associated Devices of the device's WiFi Access Points the following would be configured:

- .BulkData.Profile.1.Parameter.1.Name = "WiFi_AP_Assoc_BSent"
- .BulkData.Profile.1.Parameter.1.Reference = "Device.WiFi.AccessPoint.*.AssociatedDevice.*.Stats.BytesSent"

Using this configuration a device that has 2 WiFi Access Points (with instance identifiers 1 and 3) each with 2 Associated Devices (with instance identifiers 10 and 11), would contain a Report with following parameter names: WiFi_AP_Assoc_BSent.1.10, WiFi_AP_Assoc_BSent.1.11, WiFi_AP_Assoc_BSent.3.10, WiFi_AP_Assoc_BSent.3.11.

Partial paths for parameters can also be used to report all parameters of the associated Object Instance. When the value of the Reference parameter is a partial path, the value of the Name parameter (as used in a report) MUST reflect the remainder of the parameter path. Specifically, the value of Name parameter MUST be appended with a "." and then the remainder of the parameter path.
For example, for a device to report the statistics of a WiFi associated device object instance the following would be configured:

- .BulkData.Profile.1.Parameter.1.Name = " WiFi_AP1_Assoc10"
- .BulkData.Profile.1.Parameter.1.Reference = "Device.WiFi.AccessPoint.1.AssociatedDevice.10.Stats."

Using the configuration the device's report would contain the following parameter names:
WiFi_AP1_Assoc10.BytesSent, WiFi_AP1_Assoc10.BytesReceived,
WiFi_AP1_Assoc10.PacketsSent, WiFi_AP1_Assoc10.PacketsReceived,
WiFi_AP1_Assoc10.ErrorsSent, WiFi_AP1_Assoc10.RetransCount,
WiFi_AP1_Assoc10.FailedRetransCount, WiFi_AP1_Assoc10.RetryCount,
WiFi_AP1_Assoc10.MultipleRetryCount.

It is also possible for the value of the Reference parameter to use both wildcards for instance identifiers and be a partial path. For example, for device to report the statistics for the device's WiFi associated device, the following would be configured:

- .BulkData.Profile.1.Parameter.1.Name = "WiFi_AP_Assoc"
- .BulkData.Profile.1.Parameter.1.Reference = "Device.WiFi.AccessPoint.*.AssociatedDevice.*.Stats."

Using this configuration a device that has 1WiFi Access Point (with instance identifier 10) with 2 Associated Devices (with instance identifiers 10 and 11), would contain a Report with following parameter names:WiFi_AP_Assoc.1.10.BytesSent,
WiFi_AP_Assoc.1.10.BytesReceived, WiFi_AP_Assoc.1.10.PacketsSent,
WiFi_AP_Assoc.1.10.PacketsReceived, WiFi_AP_Assoc.1.10.ErrorsSent,
WiFi_AP_Assoc.1.10.RetransCount, WiFi_AP_Assoc.1.10.FailedRetransCount,
WiFi_AP_Assoc.1.10.RetryCount, WiFi_AP_Assoc.1.10.MultipleRetryCount,
WiFi_AP_Assoc.1.11.BytesSent, WiFi_AP_Assoc.1.11.BytesReceived,
WiFi_AP_Assoc.1.11.PacketsSent, WiFi_AP_Assoc.1.11.PacketsReceived,
WiFi_AP_Assoc.1.11.ErrorsSent, WiFi_AP_Assoc.1.11.RetransCount,
WiFi_AP_Assoc.1.11.FailedRetransCount, WiFi_AP_Assoc.1.11.RetryCount,
WiFi_AP_Assoc.1.11.MultipleRetryCount.

A.3.3 Processing of Content for Failed Report Transmissions

When the content (report) cannot be successfully transmitted, including retries, to the data collector, the NumberOfRetainedFailedReports parameter of the BulkData.Profile object instance defines how the content should be disposed based on the following rules:

- When the value of the NumberOfRetainedFailedReports parameter is greater than 0, then the report for the current reporting interval is appended to the list of failed reports. How the content is appended is dependent on the type of encoding (e.g., CSV, JSON) and is described further in corresponding encoding section.
- If the value of the NumberOfRetainedFailedReports parameter is -1, then the CPE will retain as many failed reports as possible.
- If the value of the NumberOfRetainedFailedReports parameter is 0, then failed reports are not to be retained for transmission in the next reporting interval.
- If the CPE cannot retain the number of failed reports from previous reporting intervals while transmitting the report of the current reporting interval, then the oldest failed reports are deleted until the CPE is able to transmit the report from the current reporting interval.
- If the value BulkData.Profile object instance's EncodingType parameter is modified any outstanding failed reports are deleted.

A.3.4 Encoding of CSV Bulk Data

CSV Bulk Data SHOULD be encoded as per RFC 4180 [16] and MUST contain a header line (column headers) and the media type MUST indicate the presence of the header line.
For example: **Content-Type: text/csv; charset=UTF-8; header=present**. The formatting of the header line is defined in section A.2.1.

In addition, the characters used to separate fields and rows as well as identify the escape character can be configured from the characters used in RFC-4180.

Using the HTTP example in A.2, the following configures the CPE to transfer data to the Bulk Data Collector using CSV encoding, separating the fields with a comma and the rows with a new line character, by setting the following parameters using the SetParameterValues RPC as follows:

- .BulkData.Profile.1.EncodingType = "CSV"
- .BulkData.Profile.1 CSVEncoding.FieldSeparator = ","
- .BulkData.Profile.1.CSVEncoding.RowSeparator="
"
- .BulkData.Profile.1.CSVEncoding.EscapeCharacter="""

A.3.4.1 Defining the Report Layout of the Encoded Bulk Data

The layout of the data in the reports associated with the profiles allows parameters to be formatted either as part of a column (ParameterPerColumn) or as a distinct row (ParameterPerRow) as defined in section A.4.1. In addition, the report layout allows rows of data to be inserted with a timestamp stating when the data is collected.

Using the HTTP example in A.2, the following configures the CPE to format the data using a parameter as a row and inserting a timestamp as the first column entry in each row using the "Unix-Epoch" time. The information is configured by setting the following parameters using the SetParameterValues RPC as follows:

- .BulkData.Profile.1.CSVEncoding.ReportFormat ="ParameterPerRow"
- .BulkData.Profile.1.CSVEncoding.RowTimestamp ="Unix-Epoch"

The report format of "ParameterPerRow" MUST format each parameter using the ParameterName, ParameterValue and ParameterType in that order. The ParameterType MUST be the parameter's base data type as described in TR-106 [2].

A.3.4.2 Layout of Content for Failed Report Transmissions

When the value of the NumberOfRetainedFailedReports parameter of the BulkData.Profile object instance is -1 or greater than 0, then the report of the current reporting interval is appended to the failed reports. For CSV Encoded data the content of new reporting interval is added onto the existing content without any header data.

A.3.5 Encoding of JSON Bulk Data

Using the HTTP example in A.2, a SetParameterValues RPC is used to configure the CPE to transfer data to the Bulk Data Collector using JSON encoding as follows:

- .BulkData.Profile.1.EncodingType = "JSON"

A.3.5.1 Defining the Report Layout of the Encoded Bulk Data

Reports that are encoded with JSON Bulk Data are able to utilize different report format(s) defined by the JSONEncoding object's ReportFormat parameter as defined in section A.4.2. In addition, a "CollectionTime" JSON object can be inserted into the report instance that defines when the data for the report was collected.

The following configures the CPE to encode the data using a parameter as JSON Object named "CollectionTime" using the "Unix-Epoch" time format using the SetParameterValues RPC as follows:

- .BulkData.Profile.1.JSONEncoding.ReportTimestamp ="Unix-Epoch"

Note: The encoding format of "CollectionTime" is defined as an JSON Object parameter encoded as: **"CollectionTime":1364529149**

Reports are defined as an Array of Report instances encoded as:

"Report":[{...},{...}]

Note: Multiple instances of Report instances may exist when previous reports have failed to be transmitted.

A.3.5.2 Layout of Content for Failed Report Transmissions

When the value of the NumberOfRetainedFailedReports parameter of the BulkData.Profile object instance is -1 or greater than 0, then the report of the current reporting interval is appended to the failed reports. For JSON Encoded data the report for the current reporting interval is added onto the existing appended as a new "Data" object array instance as shown below:
Reports are defined as an Array of Report instances encoded as:

```
"Report": [
        {Report from a failed reporting interval},
        {Report from the current reporting interval}
        ]
```

A.3.5.3 Using the ObjectHierarchy Report Format

When a BulkData profile utilizes the JSON encoding type and has a JSONEncoding.ReportFormat parameter value of "ObjectHierarchy", then the JSON objects are encoded such that each object in the object hierarchy of the data model is encoded as a corresponding hierarchy of JSON Objects with the parameters (i.e., parameterName, parameterValue) of the object specified as name/value pairs of the JSON Object.

For example the translation for the leaf object "**Device.MoCA.Interface.*.Stats.**" would be:

```
{
  "Report": [
    {
      "Device": {
        "MoCA": {
          "Interface": {
            "1": {
              "Stats": {
                "BroadPktSent": 25248,
                "BytesReceived": 200543250,
                "BytesSent": 25248,
                "MultiPktReceived": 200543250
              }
            },
            "2": {
              "Stats": {
                "BroadPktSent": 93247,
                "BytesReceived": 900543250,
                "BytesSent": 93247,
                "MultiPktReceived": 900543250
              }
            }
          }
        }
      }
    }
```



```
  ]
}
```

Note: The translated JSON Object name does not contain the trailing period "." of the leaf object.

A.3.5.4 Using the NameValuePair Report Format

When a BulkData profile utilizes the JSON encoding type and has a JSONEncoding.ReportFormat parameter value of "NameValuePair", then the JSON objects are encoded such that each parameter of the data model is encoded as an array instance with the parameterName representing JSON name token and parameterValue as the JSON value token.

For example the translation for the leaf object "**Device.MoCA.Interface.*.Stats.**" would be:

```
{
  "Report": [
    {
      "Device.MoCA.Interface.1.Stats.BroadPktSent": 25248,
      "Device.MoCA.Interface.1.Stats.BytesReceived": 200543250,
      "Device.MoCA.Interface.1.Stats.BytesSent": 25248,
      "Device.MoCA.Interface.1.Stats.MultiPktReceived": 200543250,
      "Device.MoCA.Interface.2.Stats.BroadPktSent": 93247,
      "Device.MoCA.Interface.2.Stats.BytesReceived": 900543250,
      "Device.MoCA.Interface.2.Stats.BytesSent": 93247,
      "Device.MoCA.Interface.2.Stats.MultiPktReceived": 900543250
    }
  ]
}
```

Note: The translated JSON Object name does not contain the trailing period "." of the leaf object.

A.3.5.5 Translating Data Types

JSON has a number of basic data types that are translated from the base data types defined in TR-106 [2]. The encoding of JSON Data Types MUST adhere to RFC 7159 [17].

TR-106 named data types are translated into the underlying base TR-106 data types.
Lists based on TR-106 base data types utilize the JSON String data type.

Table 3 – TR-106 Data Type Translation - JSON

TR-106 Data Type	JSON Data Type
base64	String: base64 representation of the binary data.
boolean	Boolean
dateTime	String represented as an ISO-8601 timestamp.
hexBinary	String: hex representation of the binary data.

TR-106 Data Type	JSON Data Type
int, long, unsignedInt, unsignedLong	Number
string	String

A.4 Report Examples

This section provides example report configurations along with the examples of how the resulting encoded data would look as it is transferred to the Bulk Data Collector.

A.4.1 CSV Encoded Report Examples

A.4.1.1 CSV Encoded Reporting Using ParameterPerRow Report Format

Using the configuration examples provided in the previous sections the configuration for a CSV encoded HTTP report using the ParameterPerRow report format:

- .BulkData.Profile.1
- .BulkData.Profile.1.Enable=true
- .BulkData.Profile.1.Protocol = "HTTP"
- .BulkData.Profile.1.ReportingInterval = 300
- .BulkData.Profile.1.TimeReference = "0001-01-01T00:00:00Z"
- .BulkData.Profile.1.HTTP.URL = "https://bdc.acme.com/somedirectory"
- .BulkData.Profile.1.HTTP.Username = "username"
- .BulkData.Profile.1.HTTP.Password = "password"
- .BulkData.Profile.1.HTTP.Compression = "Disabled"
- .BulkData.Profile.1.HTTP.Method = "POST"
- .BulkData.Profile.1.HTTP.UseDateHeader = true
- .BulkData.Profile.1.EncodingType = "CSV"
- .BulkData.Profile.1 CSVEncoding.FieldSeparator = ","
- .BulkData.Profile.1.CSVEncoding.RowSeparator="
"
- .BulkData.Profile.1.CSVEncoding.EscapeCharacter="""
- .BulkData.Profile.1.CSVEncoding.ReportFormat ="ParameterPerRow"
- .BulkData.Profile.1.CSVEncoding.ReportTimestamp ="Unix-Epoch"

- .BulkData.Profile.1.Parameter.1.Name = ""
- .BulkData.Profile.1.Parameter.1.Reference = "Device.MoCA.Interface.1.Stats.BroadPktSent"
- .BulkData.Profile.1.Parameter.2.Name = ""

- .BulkData.Profile.1.Parameter.2.Reference = "Device.MoCA.Interface.1.Stats.BytesReceived"
- .BulkData.Profile.1.Parameter.3.Name = ""
- .BulkData.Profile.1.Parameter.3.Reference = "Device.MoCA.Interface.1.Stats.BytesSent"
- .BulkData.Profile.1.Parameter.4.Name = ""
- .BulkData.Profile.1.Parameter.4.Reference = "Device.MoCA.Interface.1.Stats.MultiPktReceived"

The resulting CSV encoded data would look like:
ReportTimestamp,ParameterName,ParameterValue,ParameterType
1364529149,Device.MoCA.Interface.1.Stats.BroadPktSent,25248,unsignedLong
1364529149,Device.MoCA.Interface.1.Stats.BytesReceived,200543250,unsignedLong
1364529149, Device.MoCA.Interface.1.Stats.Stats.BytesSent,7682161,unsignedLong
1364529149,Device.MoCA.Interface.1.Stats.MultiPktReceived,890682272,unsignedLong

A.4.1.2 CSV Encoded Reporting Using ParameterPerColumn Report Format

Using the configuration examples provided in the previous sections the configuration for a CSV encoded HTTP report using the ParameterPerColumn report format:

- .BulkData.Profile.1
- .BulkData.Profile.1.Enable=true
- .BulkData.Profile.1.Protocol = "HTTP"
- .BulkData.Profile.1.ReportingInterval = 300
- .BulkData.Profile.1.TimeReference = "0001-01-01T00:00:00Z"
- .BulkData.Profile.1.HTTP.URL = "https://bdc.acme.com/somedirectory"
- .BulkData.Profile.1.HTTP.Username = "username"
- .BulkData.Profile.1.HTTP.Password = "password"
- .BulkData.Profile.1.HTTP.Compression = "Disabled"
- .BulkData.Profile.1.HTTP.Method = "POST"
- .BulkData.Profile.1.HTTP.UseDateHeader = true
- .BulkData.Profile.1.EncodingType = "CSV"
- .BulkData.Profile.1 CSVEncoding.FieldSeparator = ","
- .BulkData.Profile.1.CSVEncoding.RowSeparator="
"
- .BulkData.Profile.1.CSVEncoding.EscapeCharacter="""

- .BulkData.Profile.1.CSVEncoding.ReportFormat ="ParameterPerColumn"
- .BulkData.Profile.1.CSVEncoding.ReportTimestamp ="Unix-Epoch"

- .BulkData.Profile.1.Parameter.1.Name = "BroadPktSent"
- .BulkData.Profile.1.Parameter.1.Reference = "Device.MoCA.Interface.1.Stats.BroadPktSent"
- .BulkData.Profile.1.Parameter.2.Name = "BytesReceived"
- .BulkData.Profile.1.Parameter.2.Reference = "Device.MoCA.Interface.1.Stats.BytesReceived"
- .BulkData.Profile.1.Parameter.3.Name = "BytesSent"
- .BulkData.Profile.1.Parameter.3.Reference = "Device.MoCA.Interface.1.Stats.BytesSent"
- .BulkData.Profile.1.Parameter.4.Name = "MultiPktReceived"
- .BulkData.Profile.1.Parameter.4.Reference = "Device.MoCA.Interface.1.Stats.MultiPktReceived"

The resulting CSV encoded data with transmission of the last 3 reports failed to complete would look like:

ReportTimestamp,BroadPktSent,BytesReceived,BytesSent,MultiPktReceived
1364529149,25248,200543250,7682161,890682272
1464639150,25249,200553250,7683161,900683272
1564749151,25255,200559350,7684133,910682272
1664859152,25252,200653267,7685167,9705982277

A.4.2 JSON Encoded Report Example

Using the configuration examples provided in the previous sections the configuration for a JSON encoded HTTP report:

- .BulkData.Profile.1
- .BulkData.Profile.1.Enable=true
- .BulkData.Profile.1.Protocol = "HTTP"
- .BulkData.Profile.1.ReportingInterval = 300
- .BulkData.Profile.1.TimeReference = "0001-01-01T00:00:00Z"
- .BulkData.Profile.1.HTTP.URL = "https://bdc.acme.com/somedirectory"
- .BulkData.Profile.1.HTTP.Username = "username"
- .BulkData.Profile.1.HTTP.Password = "password"

- .BulkData.Profile.1.HTTP.Compression = "Disabled"
- .BulkData.Profile.1.HTTP.Method = "POST"
- .BulkData.Profile.1.HTTP.UseDateHeader = true
- .BulkData.Profile.1.EncodingType = "JSON"
- .BulkData.Profile.1.JSONEncoding.ReportFormat ="ObjectHierarchy"
- .BulkData.Profile.1.JSONEncoding.ReportTimestamp ="Unix-Epoch"

- .BulkData.Profile.1.Parameter.1.Reference = "Device.MoCA.Interface.*.Stats."

The resulting JSON encoded data would look like:

```
{
  "Report": [
    {
      "CollectionTime": 1364529149,
      "Device": {
        "MoCA": {
          "Interface": {
            "1": {
              "Stats": {
                "BroadPktSent": 25248,
                "BytesReceived": 200543250,
                "BytesSent": 25248,
                "MultiPktReceived": 200543250
              }
            },
            "2": {
              "Stats": {
                "BroadPktSent": 93247,
                "BytesReceived": 900543250,
                "BytesSent": 93247,
                "MultiPktReceived": 900543250
              }
            }
          }
        }
      }
    }
  ]
}
```

Note: All supported parameters for the "Device.MoCA.Interface.*.Stats." object would be encoded based on the .BulkData.Profile.1.Parameter.1.Reference =

"Device.MoCA.Interface.*.Stats." selection. This example just depicts 4 of the parameters of the referenced object.

If the value of the .BulkData.Profile.1.JSONEncoding.ReportFormat parameter was "NameValuePair", the results of the configuration are:


```
{
  "Report": [
    {
      "CollectionTime": 1364529149,
      "Device.MoCA.Interface.1.Stats.BroadPktSent": 25248,
      "Device.MoCA.Interface.1.Stats.BytesReceived": 200543250,
      "Device.MoCA.Interface.1.Stats.BytesSent": 25248,
      "Device.MoCA.Interface.1.Stats.MultiPktReceived": 200543250,
      "Device.MoCA.Interface.2.Stats.BroadPktSent": 93247,
      "Device.MoCA.Interface.2.Stats.BytesReceived": 900543250,
      "Device.MoCA.Interface.2.Stats.BytesSent": 93247,
      "Device.MoCA.Interface.2.Stats.MultiPktReceived": 900543250
    }
  ]
}
```


Appendix I. USB Host Theory of Operation

I.1 Overview

An increasing number of devices are equipped with a USB Host controller and USB host interface(s) / connector(s) (series A receptacle).

There are a number of use cases for adding a USB Host and connected devices to a CWMP data model. One example is retrieving the exact product identity of the connected device in the event of service issues such as printer or file sharing problems. Another example is notifying the user that a newly-connected device is not supported, e.g. due to a missing driver. Or the detection of the connection of a particular USB device could mean additional services for this device could be offered to the subscriber.

The data model contains the number of devices connected to each host controller. For each device, the main properties of the USB device descriptors as well as interface descriptors are represented. The latter is to support devices that only indicate class/subclass (and therefore device type) at the interface level.

Example USB topology of connected devices:

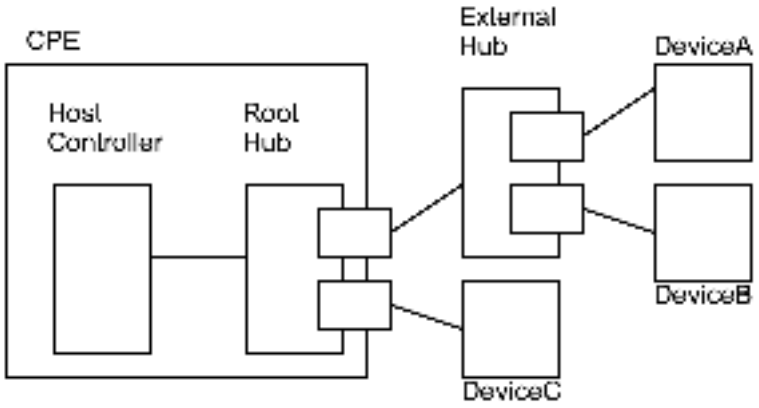

Figure 1 - Example USB Host Connections

All USB devices attach to a USB Host through a port on a USB entity known as a hub. Hubs have status bits that are used to report the attachment or removal of a USB device on one of its ports. The USB Host queries the hub to retrieve these status bits. In the case of an attachment, the USB Host enables the port and addresses the USB device through the device's control pipe at the default address. Figure 1 depicts both a Root Hub and an External Hub that provide this service.

The USB Host assigns a unique USB address to the device and then determines if the newly attached USB device is a hub or function. The USB Host establishes its end of the control pipe for the USB using the assigned USB address and endpoint number zero. This is reflected in the data model by adding a new USBHosts.Host.{i}.Device.{i}. instance.

If the attached USB device is a hub and USB devices are attached to its ports, then the above procedure is followed for each of the attached USB devices.

If the attached USB device is a function, then attachment notifications will be handled by the USB Host software that is appropriate for the function.

Appendix II. Software Module Management

This section discusses the Theory of Operation for Software Module Management using TR-069 [1] and the Software Module object defined in the Root data model.

II.1 Overview

As the home networking market matures, CPE in the home are becoming more sophisticated and more complex. One trend in enhanced device functionality is the move towards more standardized platforms and execution environments (such as Java, Linux, OSGi, etc.). Devices implementing these more robust platforms are often capable of downloading new applications dynamically, perhaps even from third-party software providers. These new applications might enhance the existing capabilities of the device or enable the offering of new services to the subscriber.

This model differs from previous CPE software architectures that assumed one monolithic firmware that was downloaded and applied to the device in one action.

That sophistication is a double-edged sword for service providers. On one hand, these devices are able to offer new services to subscribers and therefore increase the revenue per subscriber, help operators differentiate, and reduce churn with "sticky" applications that maintain subscriber interest. On the other hand, the increased complexity creates more opportunities for problems, especially as the users of these home-networking services cease to be early adopters and move into the mainstream. It is important that the increased revenue opportunity is not offset with growing activation and support costs.

In order to address the need of providing more compelling dynamic applications on the CPE while ensuring a smooth "plug and play" user experience, it is necessary for service providers to make use of CMWP to remotely manage the life cycle of these applications, including install, activation, configuration, upgrade, and removal. Doing so ensures a positive user experience, improves service time-to-market, and reduces operational costs related with provisioning, support, and maintenance.

II.2 Lifecycle Management

There are a number of actions that service providers might want to take in managing the lifecycle of these dynamic applications. They might want to install new applications for the subscriber. They might want to update existing applications when new versions or patches are available.

They might want to start and/or stop these applications as well. Finally, they might want to uninstall applications that are no longer needed (or perhaps paid for) by the subscriber.

The specifics of how applications run in different environments vary from platform to platform. In order to avoid lifecycle management tailored to each specific operating environment, CWMP-based software management defines abstract state models and abstract software module concepts as described in the following sections. These concepts are not tied to any particular platform and enable CWMP to manage dynamic software on a wide range of devices in a wide range of environments.

II.3 Software Modules

A **Software Module** is any software entity that will be installed on a CPE. This includes modules that can be installed/uninstalled and those that can be started and stopped. All software on the device is considered a software module, with the exception of the primary firmware, which plays a different enough role that it is considered a separate entity.

A software module exists on an **Execution Environment** (EE), which is a software platform that supports the dynamic loading and unloading of modules. It might also enable the dynamic sharing of resources among entities, but this differs across various execution environments. Typical examples include Linux, OSGi, .NET, Android, and Java ME. It is also likely that these environments could be "layered," i.e., that there could be one primary environment such as Linux on which one or more OSGi frameworks are stacked. This is an implementation specific decision, however, and CWMP-based module management does not attempt to enable management of this layering beyond exposing which EE a given environment is layered on top of (if any). CWMP-based Software Module Management also does not attempt to address the management of the primary firmware image, which is expected to be managed via the Download mechanism previously defined in TR-069.

Software modules come in two types: **Deployment Units** (DUs) and **Execution Units** (EUs). A DU is an entity that can be deployed on the EE. It can consist of resources such as functional EUs, configuration files, or other resources. Fundamentally it is an entity that can be Installed, Updated, or Uninstalled. Each DU can contain zero or more EUs but the EUs contained within that DU cannot span across EEs. An EU is an entity deployed by a DU, such as services, scripts, software components, or libraries. The EU initiates processes to perform tasks or provide services. Fundamentally it is an entity that can be Started or Stopped. EUs also expose configuration for the services implemented, either via standard TR-069 related data model objects and parameters or via EU specific objects and parameters.

It is possible that Software Modules can have dependencies on each other. For example a DU could contain an EU that another DU depends on for functioning. If all the resources on which a DU depends are present and available on an EE, it is said to be Resolved. Otherwise the EUs associated with that DU might not be able to function as designed. It is outside the scope of Software Module Management to expose these dependencies outside of indicating whether a particular DU is RESOLVED or not.

II.3.1 Deployment Units

Below is the state machine diagram[2] for the lifecycle of DUs.

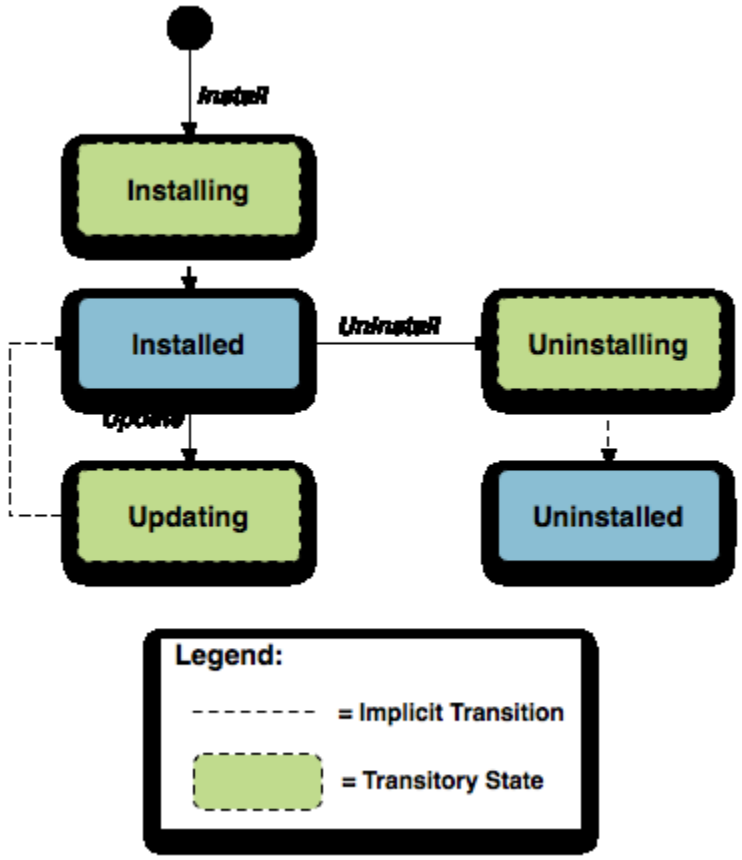

Figure 2 – Deployment Unit State Diagram

This state machine shows 5 individual states (3 of which are transitory) and 3 explicitly triggered state transitions.

The explicit transitions among the non-transitory states are triggered by a CMWP method call, ChangeDUState, defined in Section A.4.1.10 / TR-069 [1]. The explicit transitions are as follows:

1. Install, which initiates the process of Installing a DU. The device might need to transfer a file from the location indicated by a URL in the method call. Once the resources are available on the device, the CPE begins the installation process:
   - In the Installing state, the DU is in the process of being Installed and will transition to that state unless prevented by a fault. Note that the ACS has the option to choose which EE to install a particular DU to, although it can also leave that choice up to the CPE. If the ACS does specify the EE, it is up to the ACS to specify one that is

[2] This state machine diagram refers to the successful transitions caused by the ChangeDUState RPC and does not model the error cases.

compatible with the DU it is attempting to Install (e.g., an OSGi framework for an OSGi bundle).

- In the Installed state, the DU has been successfully downloaded and installed on the relevant EE. At this point it might or might not be Resolved. If it is Resolved, the associated EUs can be started; otherwise an attempt to start the associated EUs will result in a failure. How dependencies are resolved is implementation and EE dependent.

2. Update, which initiates a process to update a previously existing DU. As with Install, the device might need to transfer a file from the location indicated by a URL in the method call. If no URL is provided in the request, the CPE uses the last URL stored in the DeploymentUnit table (including any related authentication credentials) used from either Install or a previous Update. There are four combinations of URL and UUID being supplied in the request; for details, see Section A.4.1.10 in TR-069 [1]. Once the resources are available on the device, the CPE begins the updating process:

- In the Updating state, the DU is in the process of being Updated and will transition to the Installed state. As with initial installation, the DU might or might not have dependencies Resolved at this time.
- During the Updating state, the associated EUs that had been in the Active state transition to Idle during the duration of the Update. They are automatically restarted once the Update process is complete.

Note that an Update is performed on the underlying resource(s) across all EEs with which the DU is associated. Each affected DU instance, however, has its own result entry in the DUStateChangeComplete method.

3. Uninstall, which initiates the process of uninstalling the DU and removing the resources from the device. It is possible that a DU to be Uninstalled could have been providing shared dependencies to another DU; it is possible therefore that the state of other DUs and/or EUs could be affected by the DU being Uninstalled.

- In the Uninstalling state, the DU is in the process of being Uninstalled and will transition to that state unless prevented by a fault.
- In the Uninstalled state, the DU is no longer available as a resource on the device. Garbage clean up of the actual resources are EE and implementation dependent. In many cases, the resource(s) will be removed automatically at the time of un-installation. The removal of any associated EUs is part of DU clean up.

The ChangeDUState method can contain any combination of requested operations over independent multiple DUs. Because the CPE is allowed to apply the operations in any order of its choosing (even though it needs to report the results in the order received in the request) the ACS cannot depend on operations being deployed in a specific order to a given DU; this means that if an ACS wants to perform ordered operations on a specific DU, it needs to do so in multiple method calls. CPE are required to accept at least 16 operations in a method call; there is no theoretical upper bound on the number of operations that can be triggered in a single ChangeDUState method, but it is limited by the resources and capabilities of the device itself. The ChangeDUState method is an asynchronous request, meaning that, except in cases where the

request fails, the CPE notifies the ACS in a subsequent CWMP session about the success or failure of the state transitions requested using a DUStateChangeComplete ACS method (see below for more information on fault scenarios).
These state transitions might also be triggered via means other than CWMP (e.g. user-triggered or CPE-triggered). Since the ACS might still be interested in knowing about these autonomous state changes there is also an ACS method, called AutonomousDUStateChangeComplete, for this purpose. The ACS can filter the notifications it receives via this mechanism using the parameters defined in the ManagementServer.DUStateChangeComplPolicy object.

The inventory of available DUs along with their current state can be found in the SoftwareModules component found in the Root data model, i.e., the SoftwareModules.DeploymentUnit.{i} object. This object contains a list of all the DUs currently on the device, along with pertinent information such as DU identifiers, current state, whether the DU is Resolved, information about the DU itself such as vendor and version, the list of associated EUs, and the EEs on which the particular DU is installed.

DUs have a number of identifiers, each contributed by a different actor in the ecosystem:

- A Universally Unique Identifier (UUID) either assigned by the management server (ACS) or generated by the CPE at the time of Installation. . This identifier gives the management server a means to uniquely identify a particular DU across the population of devices on which it is installed. A DU will, therefore, have the same UUID on different devices, but there can be no more than one DU with the same UUID and version installed to an EE on a particular device. See II.3.1.1 below for more information on UUID generation.
- A Deployment Unit Identifier (DUID) assigned by the EE on which it is deployed; this identifier is specific to the particular EE, and different EEs might have different logic for the assigning of this value.
- A Name assigned by the author of the DU.

The creation of a particular DU instance in the data model occurs during the Installation process. It is at this time that the DUID is assigned by the EE. Upon Uninstall, the data model instance will be removed from the DU table once the resource itself has been removed from the device. Since garbage clean up is EE and implementation dependent, it is therefore possible that a particular DU might never appear in the data model in the Uninstalled state but rather disappear at the time of the state transition. It is also possible that an event, such as a Reboot, could be necessary before the associated resources are removed.

II.3.1.1 UUID Generation

An important aspect of the UUID is that it might be generated by either the ACS and provided to the CPE as part of the Install operation, or generated by the CPE either if the ACS does not provide a UUID in the Install operation or if the DU is Installed outside CWMP-based management, such as at the factory or via a LAN-side mechanism (e.g. UPnP DM). Because the UUID is meant to uniquely identify a DU across a population of devices, it is important that the UUID be the same whether generated by the ACS or the CPE. In order to ensure this, the UUID is generated (whether by ACS or CPE) according to the rules defined by RFC 4122 [7] Version 3 (Name-Based) and Annex H / TR-069 [1]. The following are some possible scenarios:

1. The DU is Installed via CWMP with an ACS generated UUID and is subsequently Updated/Uninstalled via CWMP. All post-Install management actions require the UUID to address the DU, which is retained across version changes.

2. The DU is factory Installed with a CPE generated UUID and is subsequently Updated/Uninstalled via CWMP. In this case the ACS can either choose to generate this UUID if it has access to the information necessary to create it or to learn the UUID by interrogating the data model.

3. The DU is Installed via CWMP with an ACS generated UUID and is subsequently Updated/Uninstalled via a LAN-side mechanism. In this scenario it is possible that the LAN-side mechanism is unaware of the UUID and uses its own protocol-specific mechanism to identify and address the DU. The UUID, however, is still retained across version changes. If AutonomousDUStateChangeComplete notifications are enabled for the device, the CPE also sends that method (containing the UUID) to the ACS once the LAN-side triggered state change has completed.

4. The DU is Installed via CWMP but the ACS provides no UUID in the Install operation. In this case the CPE generates the UUID, which must be used by the ACS in any future CWMP-based Updates or Uninstalls. Depending on its implementation, the ACS might choose to generate the UUID at the time of the future operations, learn the value of the UUID from the DUStateChangeComplete RPC, or learn it by interrogating the data model.

5. The DU is Installed via a LAN-side mechanism and is subsequently Updated/Uninstalled via CWMP. Since it is likely that the LAN-side mechanism does not provide a Version 3 Name-Based UUID in its protocol-specific Install operation, it is expected that the CPE generates the UUID in this case when it creates the DU instance in the data model. Depending on its implementation, the ACS might choose to generate the UUID for later operations if it has access to the information necessary to create it, learn the UUID from the AutonomousDUStateChangeComplete RPC (if this notification mechanism is enabled), or learn it by interrogating the data model.

II.3.2 Execution Units

Below is the state machine diagram[3] for the lifecycle of EUs.

[3] This state machine diagram refers to the successful transitions caused by the RequestedState Parameters within the ExecutionUnit table and does not model the error cases.

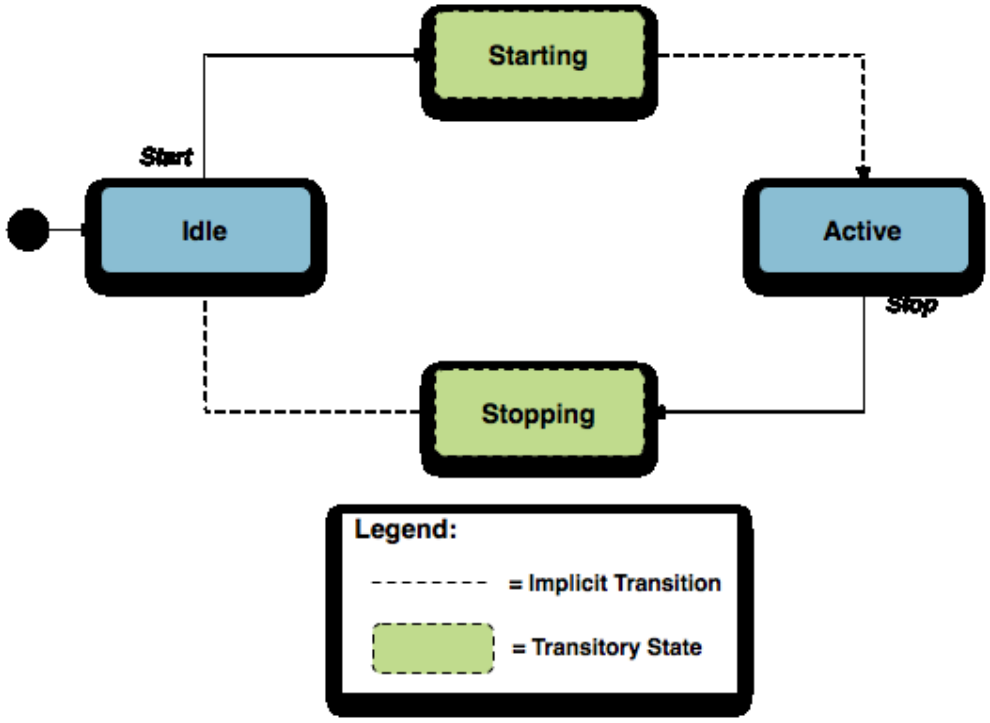

Figure 3 – Execution Unit State Diagram

This state machine shows 4 states (2 of them transitory) and two explicitly triggered state transitions.

The state transitions between the non-transitory states are triggered using the SetParameterValues method call as defined in Section A.3.2.1 / TR-069 [1] and the SoftwareModules.ExecutionUnit.{i}.RequestedState parameter as defined in the SoftwareModules object in the Root data model. The explicit transitions are as follows:

1. In order to Start an EU, the ACS sets the value of the RequestedState parameter to Active. The EU enters the Starting state, during which it takes any necessary steps to move to the Active state, and it will transition to that state unless prevented by a fault. Note that an EU can only be successfully started if the DU with which it is associated has all dependencies Resolved. If this is not the case, then the EU's status remains as Idle, and the ExecutionFaultCode and ExecutionFaultMessage parameters are updated appropriately.

2. In order to Stop an EU, the ACS sets the value of the RequestedState parameter to Idle**.** The EU enters the Stopping state, during which it takes any necessary steps to move to the Idle state, and then transitions to that state.

It is also possible that the EU could transition to the Active or Idle state without being explicitly instructed to do so by the ACS (e.g., if the EU is allowed to AutoStart, in combination with the run level mechanism, or if operation of the EU is disrupted because of a later dependency error).

The ACS manages being notified of these autonomous state changes via Active Notification on the SoftwareModules.ExecutionUnit.{i}.Status parameter. Note that this parameter is defined as having Active Notification enabled by default.

The inventory of available EUs along with their current state can be found in the SoftwareModules component found in the Root data model; i.e., the SoftwareModules.ExecutionUnit.{i} object. This object contains a list of all the EUs currently on the device along with accompanying status and any current errors as well as resource utilization related to the EU, including memory and disk space in use.

EUs have a number of identifiers, each contributed by a different actor in the ecosystem:

- An Execution Unit Identifier (EUID) assigned by the EE on which it is deployed; this identifier is specific to the particular EE, and different EEs might have different logic for assigning this value. There can be only one EU with a particular EUID.
- A Name provided by the developer and specific to the associated DU.
- A Label assigned by the EE; this is a locally defined name for the EU.

The creation of a particular EU instance in the data model occurs during the Installation process of the associated DU. It is at this time that the EUID is assigned by the EE as well. The configuration exposed by a particular EU is available from the time the EU is created in the data model, whether or not the EU is Active. Upon Uninstall of the associated DU, it is expected that the EU would transition to the Idle State, and the data model instance would be removed from the EU table once the associated resources had been removed from the device. Garbage clean up, however, is EE and implementation dependent.

Although the majority of EUs represent resources such as scripts that can be started or stopped, there are some inert resources, such as libraries, which are represented as EUs. In this case, these EUs behave with respect to the management interface as a "regular" EU. In other words, they respond successfully to Stop and Start commands, even though they have no operational meaning and update the SoftwareModules.ExecutionUnit.{i}.Status parameter accordingly. In most cases the Status would not be expected to transition to another state on its own, except in cases where its associated DU is Updated or Uninstalled or its associated EE is Enabled or Disabled, in which cases the library EU acts as any other EU.

The EUs created by the Installation of a particular DU might provide functionality to the CPE that requires configuration by the ACS. This configuration could be exposed via the CWMP data model in five ways:

1. Service data model (if, for example, the EU provides VoIP functionality, configuration would be exposed via the Voice Service data model defined in TR-104 [6]).
2. Standard objects and parameters in the device's root data model (if, for example, the EU provides port mapping capability, the configuration would be exposed via the port mapping table defined in TR-098 [5] or TR-181 Issue 2 [4]).
3. Instances of standard objects in the Root or any Service data model, (if, for example, the EU provides support for an additional Codec in a VoIP service).
4. Vendor extension objects and parameters that enhance and extend the capabilities of standard objects (if, for example, the EU provides enhanced UserInterface capabilities)

5. Standalone vendor extension objects that are directly controlled objects of the EU (for example, a new vendor specific object providing configuration for a movies on demand service).

In the case of 1 or 3, the References parameter in the EU object provides a list of path names to the services and multi-instance objects that are the directly controlled objects of the EU and which came into existence because of this particular EU. In the case of 5, the Extensions sub-object within the EU object provides a place to place these vendor extensions to allow multiple EUs to expose parameters without concern of conflicting parameter names. In the case of 2 or 4, these can be discovered using the SupportedDataModelList parameter and its links to the Current Data Model table as discussed below or through interrogation of the data model using the GetParameterNames RPC.

The creation of these additional data model objects and parameters means that the Current Supported Data Model of the device is also updated. The EU object contains a parameter that is a path reference to an instance in the SupportedDataModel table in the root data model so that the ACS can retrieve the DT file associated with the EU in order to discover its manageable characteristics.

All data model services, objects, and parameters related to a particular EU come into existence at the time of Installation or Update of the related DU, The related data model disappears from the device's data model tree at the time of Uninstall and clean up of the related DU resources. It is possible that the device could encounter errors during the process of discovering and creating EUs; if this happens, it is not expected that the device would roll back any data model it has created up until this point but would rather set the ExecutionFaultCode of the EU to "Unstartable." In this case, it is not expected that any faults (with the exception of System Resources Exceeded) would have been generated in response to the Install or Update operation. See below for more information on EU faults.

The configuration of EUs could be backed up and restored using vendor configuration files. The EU object in the data model contains a parameter, which is a path reference to an instance in the vendor config file table in the Root data model. This path reference indicates the vendor config file associated with the configuration of the particular EU. Retrieval and downloading of vendor config files occurs via the Upload and Download methods defined in TR-069 [1], just as with any config files.

It is also possible that applications could have dedicated log files. The EU object also contains a parameter, which is a path reference to an instance in the log file table in the root data model. This path reference indicates the log file associated with a particular EU. Retrieval of log files is accomplished using the Upload method as defined in TR-069 [1].

II.3.3 Example Sequence Diagrams

The following diagrams provide an example sequence for the deployment of a new Software Module, including the installation of the DU and the configuration and starting of an EU.

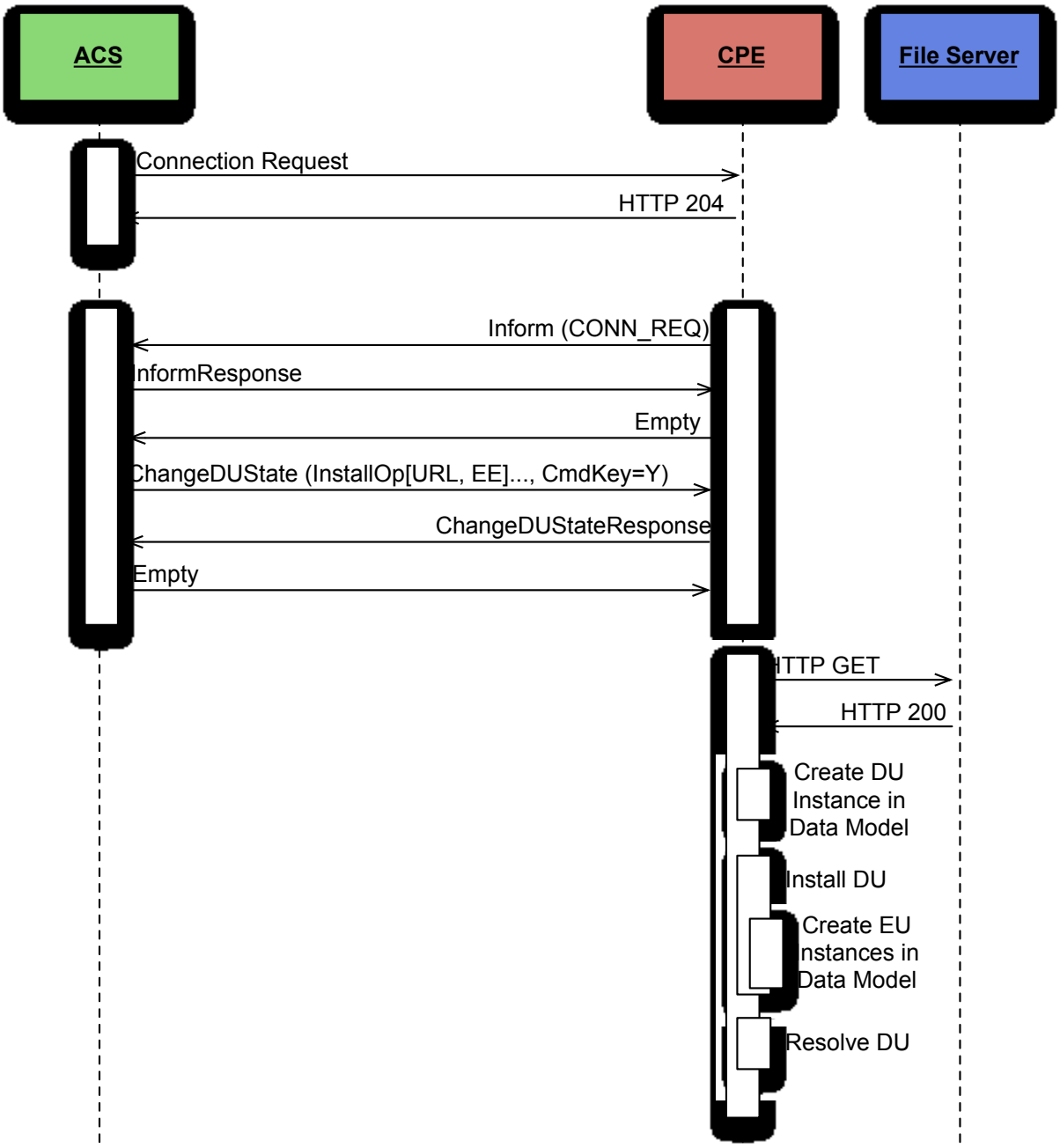

Figure 4 – Installation of a Deployment Unit - CWMP Session #1

In this first CWMP Session we see the ACS requesting an Installation of a specific Deployment Unit by providing a URL in the ChangeDUState RPC. The CPE will retrieve the file, create the Deployment Unit instance, install the Deployment Unit, create any Execution Unit instances, and finally attempt to resolve any Deployment Unit dependencies.

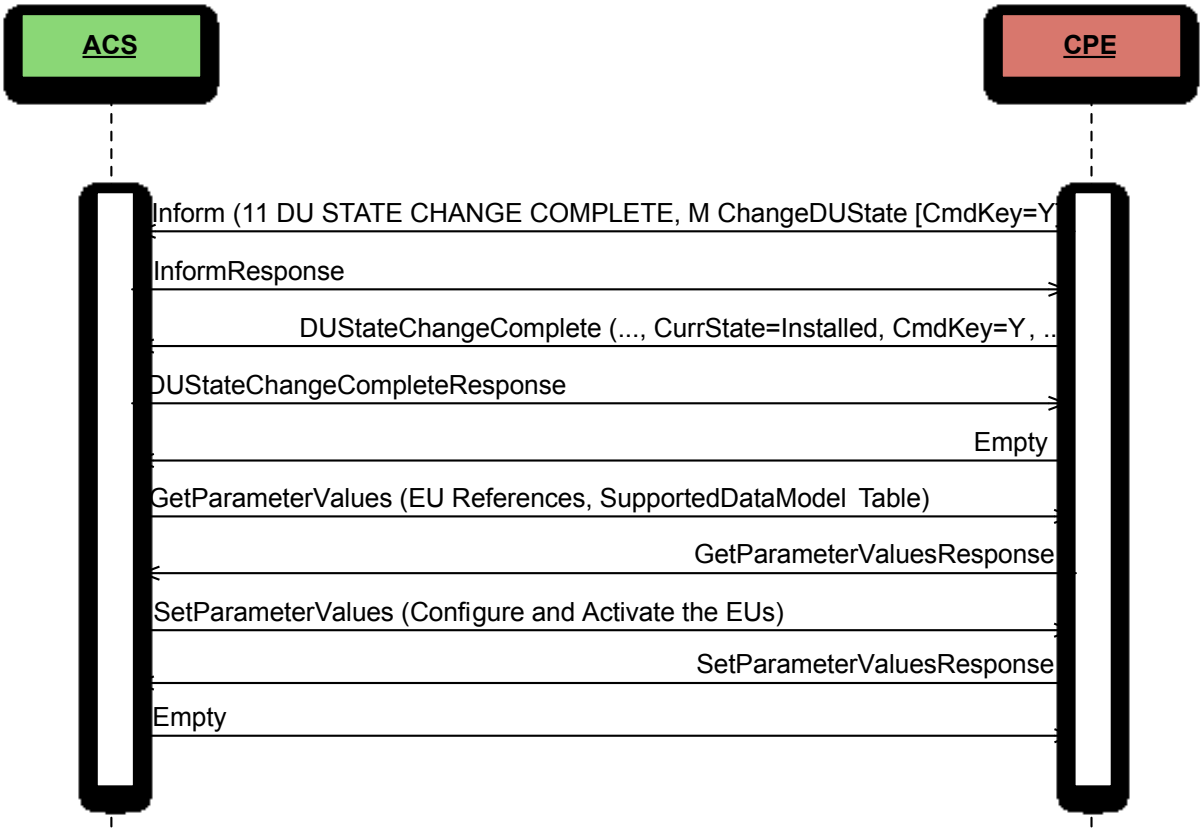

Figure 5 – Configuring and Starting the Execution Units - CWMP Session #2

In this second CWMP Session we see the CPE informing the ACS that the Deployment Unit has been successfully installed. At this point the ACS queries the Execution Unit instances that were reported back in the DUStateChangeComplete RPC so the ACS can determine what needs to be configured before activating the Execution Units. The ACS then configures the Execution Unit instances and activates them, using the RequestedState parameter with a value of "Active", within the same SetParameterValues RPC.

II.4 Execution Environment Concepts

As discussed above, an EE is a software platform that supports the dynamic loading and unloading of modules. A given device can have multiple EEs of various types and these EEs can be layered on top of each other. The following diagram gives a possible implementation of multiple EEs.

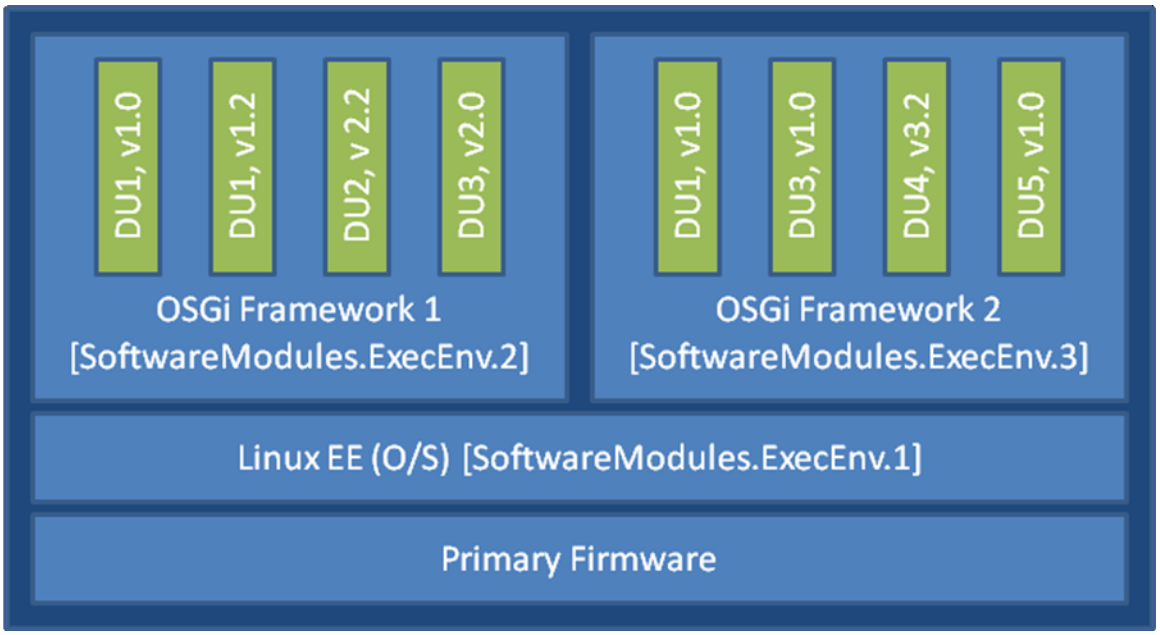

Figure 6 – Possible Multi-Execution Environment Implementation

In this example, the device exposes its Linux Operating System as an EE and has two different OSGi frameworks layered on top of it, all of which are modeled as separate ExecEnv object instances. In order to indicate the layering to the ACS, the two OSGi framework objects (.ExecEnv.2 and .ExecEnv.3) would populate the Exec.Env.{i}.Parent parameter with a path reference to the Linux object (.ExecEnv.1). The Linux EE object would populate that parameter with an empty string to indicate that it is not layered on top of any managed EE.

Multiple versions of a DU can be installed within a single EE instance, but there can only be one instance of a given version at a time. In the above diagram, there are two versions of DU1, v1.0 and v1.2 installed on .ExecEnv.2. If an attempt is made to update DU1 to version 1.2, or to install another DU with version 1.0 or 1.2, on ExecEnv.2, the operation will fail.

A DU can also be installed to multiple EEs. In the above example, DU1 is installed both to ExecEnv.2 and ExecEnv.3. The Installation is accomplished by having two different Install Actions in the ChangeDUState method call; note that it is possible for an Install to be successful on one EE and not the other or for the DU to be Resolved on one EE and not the other in this case.

When DUs are Updated, the DU instances on all EEs are affected. For example, in the above diagram, if DU1 v.1.0 is updated to version 2.0, the instances on both .ExecEnv.2 and .ExecEnv.3 will update to version 2.0.

For Uninstall, an ACS can either indicate the specific EE from which the DU should be removed, or not indicate a specific EE, in which case the DU is removed from all EEs.

An EE can be enabled and disabled by the ACS. Reboot of an EE is accomplished by first disabling and then later enabling the EE. When an EE instance is disabled by the ACS, the EE itself shuts down. Additionally, any EUs associated with the EE automatically transition to Stopped and the ExecutionFaultCode parameter value is "Unstartable." The state of the associated DUs remains the same. If a ChangeDUState method is attempted on any of the DUs associated with a disabled EE, the operation fails and an error is returned in the fault struct of the DUStateChangeComplete RPC. If an attempt is made to Start an EU associated with a Disabled EE, the device returns a CWMP fault that contains a SetParameterValues fault element for RequestedState. It should be noted if the Operating System of the device is exposed as an EE, disabling it could result in the device being put into a non-operational and non-manageable state. It should also be noted that disabling the EE on which the CWMP Management agent resides can result in the device becoming unmanageable via TR-069.

Note that the above is merely an example; whether a device supports multiple frameworks of the same type and whether it exposes its Operating System as an Execution Environment for the purposes of management is implementation specific.

II.5 Fault Model

Faults can occur at a number of steps in the software module process. The following sections discuss Deployment Unit faults and Execution Unit faults.

II.5.1 DU Faults

There are two basic types of DU faults: Operation failures and CWMP faults. CWMP faults come as a response to the ChangeDUState RPC itself; because of the atomic nature of CWMP methods, the entire method fails and none of the Operations included in the RPC are attempted. Operation failures are those faults that are reported in the FaultStruct of the DUStateChangeComplete method. Because the results RPC enables reporting of faults on each Operation, it is possible for one Operation to fail and another to execute successfully.

II.5.1.1 Install Faults

Most Install faults will be recognized before resources or instances are created on the device. When there is an Operation failure at Install, there are no resources installed on the device and no DU (or EU) instances are created in the data model. Similarly, if there are any Operation failures, besides System Resources Exceeded, there are no resources installed on the device and no DU (or EU) instances created in the data model.

The CWMP Faults defined for Install (Method Not Supported, Request Denied, and Internal Error) are general errors supported by most RPCs. One special CWMP fault to note is the Resources Exceeded error, which is used when there are too many Operations specified in the request. This error is not used to indicate that the DU has insufficient resources to support the DU file itself; this is rather indicated by the System Resources Exceeded fault discussed below. The Resources Exceeded error is not a valid error if 16 or fewer Operations are requested.

There are a number of Operation failures defined for Installation. The first category is those faults associated with the file server or attempt to transfer the DU resource and are the same as those defined for the existing Download method. These include:

- Userinfo element being specified in the URL
- The URL being unavailable (either because the host cannot be reached or because the resource is unavailable)
- Authentication failures due to incorrectly supplied credentials
- The URL transport method specified not being supported by the CPE or server
- The file transfer being interrupted (because of a device reboot or loss of connectivity, for example)

The second category of faults relate to issues with the DU and the Execution Environment. These are specific to Software Module Management and include:

- The EE reference specified by the ACS in the request not existing in the data model. Note that the ACS can simply omit the EE reference in the request and allow the CPE to choose the destination.
- The EE being disabled. This fault can occur when the request explicitly specifies a disabled EE. If there is no EE specified in the request, this fault could occur because the only possible destination EE for the DU (the only OSGi framework instance in the case of an OSGi bundle, for example) is disabled. The CPE is expected to make every attempt not to use a disabled EE in this scenario, however.
- Any mismatch existing between the DU and the EE (attempting to install a Linux package on an OSGi framework instance, for example). This fault can occur when the request explicitly specifies a mismatching EE. If there is no EE specified in the request, this fault could occur when there is no EE at all on the device that can support the DU.
- A DU of the same version already existing on the EE.

Finally there are a number of faults related to the DU resource itself. These include:

- The UUID in the request not matching the format specified in RFC 4122 [7] version 3 (Name-based).
- A corrupted DU resource, or the DU not being installable for other reasons, such as not being signed by any trusted entity
- The installation of the DU requiring more system resources, such as disk space, memory, etc., than the device has available. Note that this error is not to be used to indicate that more operations have been requested than the device can support, which is indicated by the Resourced Exceeded CWMP fault (described above).

II.5.1.2 Update Faults

When there is a fault on an Update operation of any kind, either CWMP or Operation failure, the DU remains at the version it was before the attempted DU state change, and it also remains in the Installed state (i.e., it is not Uninstalled). If for any reason the ACS wishes to remove a DU after an unsuccessful Update, it must do so manually using an Uninstall operation in the ChangeDUState method. When there is a CWMP fault at Update, there are no new resources installed on the device and no DU (or EU) instances are changed in the data model. Similarly, if there are any Operation failures, besides System Resources Exceeded, there are no new resources

installed on the device and no DU (or EU) instances are changed in the data model. The state of any associated EUs or any dependent EUs in the event of an Update failure is EE and implementation dependent.

As with Install, the CWMP Faults defined for Update (Method Not Supported, Request Denied, and Internal Error) are general errors supported by most RPCs. One special CWMP fault to note is the Resources Exceeded error, which is used when there are too many Operations specified in the request. This error is not used to indicate that the DU has insufficient resources to support the DU file itself; this is rather indicated by the System Resources Exceeded fault discussed below. The Resources Exceeded error is not a valid error if 16 or fewer Operations are requested.

There are a number of Operation failures defined for Update. The first category is those faults associated with the file server or attempt to transfer the DU resource and are the same as those defined for the existing Download method. These include:

- Userinfo element being specified in the URL
- The URL being unavailable (either because the host cannot be reached or because the resource is unavailable)
- Authentication failures due to incorrectly supplied credentials
- The URL transport method specified not being supported by the CPE or server
- The file transfer being interrupted (because of a device reboot or loss of connectivity, for example)

The second category of faults relate to issues with the DU and the Execution Environment. These are specific to Software Module Management and include:

- The EE on which the targeted DU resides being disabled. This fault can occur when the request explicitly specifies the UUID of a DU on a disabled EE or when the request explicitly specifies a URL last used by a DU on a disabled EE. If neither the URL nor UUID was specified in the request, this fault can occur when at least one DU resides on a disabled EE.
- Any mismatch existing between the DU and the EE. This fault occurs when the content of the updated DU does not match the EE on which it resides (for example, an attempt is made to Update a Linux package with a DU that is an OSGi bundle).
- Updating the DU would cause it to have the same version as a DU already installed on the EE.
- The version of the DU not being specified in the request when there are multiple versions installed on the EE.

Note that Updates are atomic across all the EEs with which a DU resource is associated; i.e., an Update is either successful across all EEs or unsuccessful across all EEs. For example, if an attempt is made to Update a DU which is installed to 2 EEs, one enabled and one disabled, the Update operation will fail for both. In this case, there would be 2 entries in the DUStateChangeComplete Results array both showing an operation failure with the same FaultCode and FaultString. In other words, the CPE would indicate that the failure occurred because of a disabled EE, even for the DU instance residing on the enabled one.

Finally there are a number of faults related to the DU resource itself. These include:

- The UUID in the request not matching the format specified in RFC 4122 [7] Version 3 (Name- Based).
- A corrupted DU resource, or the DU not being installable for other reasons, such as not being signed by any trusted entity
- The DU cannot be found in the data model. This fault can occur when the request explicitly specifies the UUID (or combination of UUID and version) of a DU that is unknown. It can also occur when the request does not specify a UUID but explicitly specifies a URL that has never been used to previously Install or Update a DU.
- Attempting to downgrade the DU version.
- Attempting to Update a DU not in the Installed state.
- Updating the DU requiring more system resources, such as disk space, memory, etc., than the device has available. Note that this error is not to be used to indicate that more operations have been requested than the device can support, which is indicated by the Resourced Exceeded CWMP fault (described above).

II.5.1.3 Uninstall Faults

When there is an Uninstall fault of any kind, either CWMP or Operation failure, the DU does not transition to the Uninstalled state and no resources are removed from the device. No changes are made to the EU-related portions of the data model (including the EU objects themselves and the related objects and parameters that came into existence because of this DU).

As with Install and Update, the CWMP Faults defined for Uninstall (Method Not Supported, Request Denied, and Internal Error) are general errors supported by most RPCs. One special CWMP fault to note is the Resources Exceeded error, which is used when there are too many Operations specified in the request.

There are three Operation failures defined for Uninstall. They are as follows:

- The EE on which the targeted DU resides is disabled. Note that if the Uninstall operation targets DUs across multiple EEs, this fault will occur if at least one of the EEs on which the DU resides is disabled.
- The DU cannot be found in the data model. If the EE is specified in the request, this error occurs when there is no UUID (or UUID and version) matching the one requested for the specified EE. If there is no EE specified in the request, this error occurs when there is no UUID matching the one in the requested on any EE in the data model, or, if the version is also specified in the request, then this error occurs when there is no DU with this combination of UUID and version on any EE in the data model.
- The UUID in the request not matching the format specified in RFC 4122 [7] Version 3 (Name- Based).
- The DU caused an EE to come into existence on which at least 1 DU is Installed.

II.5.2 EU Faults

EU state transitions are triggered by the SetParameterValues RPC. One type of EU fault is a CWMP fault sent in response to SetParameterValues. The CWMP faults defined are therefore

simply a subset of the errors defined for the generic SetParameterValues: Request Denied, Internal Error, Invalid Arguments, Invalid Parameter Name, Invalid Parameter Type, and Invalid Parameter Value.

Note that there is one case specific to Software Module Management: if the ACS tries to Start an EU on a disabled EE, the device returns a CWMP fault to that request. In this case the CPE indicates the reason behind this fault by using 9007 in the SetParameterValuesFault structure.

Because of the atomic nature of CWMP methods, if any parameter is in error in a SetParameterValues request, the entire method fails and none of the requested changes are made.

There are also Software Module Management specific faults indicated in the ExecutionFaultCode and ExecutionFaultMessage parameters in the data model. In addition to providing software module specific fault information, this parameter is especially important in a number of scenarios:

1. Asynchronous errors in the EU state transition. For example, it is possible that the CPE needs to do actions such as dependency checking that require more time than available in the context of a CWMP session. In this case it is possible that the device responds successfully to the SetParameterValues request, but later indicates that the EU is in error, with the Error Code Dependency Failure. There is also no expectation that the CPE would retry any EU state transitions triggered by a SetParameterValues request; i.e., if a device responds successfully to the CWMP request to Start an EU, but later finds the EU in error, the CPE would not attempt to retry Starting the EU.
2. Errors that occur at a later date than the original CWMP request, such as a Dependency Failure that occurs several days after successful Start of an EU because a DU providing dependencies is later Uninstalled.
3. State transition errors that are triggered by the Autostart/Run level mechanism.
4. "Autonomous" state transitions triggered outside the purview of CWMP, such as by a LAN-side protocol.

The faults in the ExecutionFaultCode parameter are defined as follows:

- FailureOnStart – the EU failed to start despite being requested to do so by the ACS.
- FailureOnAutoStart – the EU failed to start when enabled to do so automatically.
- FailureOnStop – the EU failed to stop despite being requested to do so by the ACS.
- FailureWhileActive – an EU that had previously successfully been started either via an explicit transition or automatically later fails.
- DependencyFailure – this is a more specific fault scenario in which the EU is unable to start or stops at a later date because of unresolved dependencies
- Unstartable – some error with the EU resource, its configuration, or the state of the associated DU or EE, such as the EE being disabled, prevents it from being started.

When the EU is not currently in fault, this parameter returns the value NoFault. The ExecutionFaultMessage parameter provides additional, implementation specific information about the fault in question.

The ExecutionFaultCode and ExecutionFaultMessage parameters are triggered parameters. In other words, it is not expected that an ACS could read this parameter before issuing a SetParameterValues request and see that there was a Dependency Failure that it would attempt to resolve first.

Notifications are used if the ACS wants to be notified of ongoing changes to the EU's error state.

Appendix III. Location Management

This section discusses the Theory of Operation for Location Management using TR-069 [1] and the Location object defined in the <rootobject>.DeviceInfo data model.

III.1 Overview

The Location object defined in this document is a multi-instance object that can be used by any device that needs to be able to acquire and/or express its "location."
This Location object is a multi instance object to account for the fact that a Device can acquire location information in more than one way. Location info can be acquired by:

- GPS/AGPS, i.e. provided by specific on-board circuitry such as GPS or AGPS;
- Manual, i.e. manually configured via the Device local GUI
- External, i.e. remotely configured via a number of protocols, including e.g. TR-069

Location objects can be created autonomously by the device, based on the location information it receives or by CWMP. When the Location object is created autonomously by the device, the device itself will fill the DataObject parameter with location data coming from GPS/AGPS, local GUI or an external protocol (not CWMP). When created by CWMP, it is up to the CWMP protocol to configure the DataObject parameter. Regardless of how a Location object is created, the device is responsible for populating the values of all of the location metadata (i.e., all parameters except the DataObject that contains the location information and the AcquiredTime) not writable by any external mechanism.
When a Location object is updated, the object can only be updated through the same mechanism that created it. The device will update the AcquiredTime as necessary and place the updated location data in the DataObject.
When a Location object is deleted, the object can only be deleted through the same mechanism that created it.

III.2 Multiple Instances of Location Data

Devices that need to make use of location data will need to have rules around how to deal with multiple instances of location data. These rules are out of scope for TR-069 and the proposed data model. These rules can need to be specific to a particular application. For example, if a VoIP device chooses to send location data in a SIP message, the device can include all of the instances of DataObject in that message, order the Locations Objects according to the acquisition date and time (parameter AcquiredDateTime, most recent is first) or order the Location objects according to some sort of protocol preference, such as GPS, AGPS, DHCP, HELD, TR-069, and then all others according to acquisition date and time.

A Femtocell Access Point (FAP) with multiple sources of location can also need rules for use of the Location object. If it must make decisions locally based on location, the FAP will need rules to determine the preferred location. If the FAP must send its location elsewhere, the FAP will need rules to determine whether the FAP sends all of its location data, or selects certain locations based on specific criteria.

III.3 TR-069, Manual, GPS, and AGPS Configured Location

As noted in the description of the TR-069 parameter <rootObject>.Location.{i}.DataObject., Manual, GPS, and AGPS mechanisms are formatted by the device according to the following formats specified by the IETF. An ACS that is creating an External:CWMP location will also use one of these formats:

1. Geographical coordinates formatted according to the XML syntax specified in IETF RFC5491[8] (update of RFC4119[10])
2. Civic addresses according to the XML syntax specified in IETF RFC5139 [9] (update of RFC4119[10])

Location information in these IETF RFCs is specified within the IETF framework of presence information. While these IETF RFCs specify presence information different from the Location component model assumed in the TR-069 framework, the IETF data format is adopted by BBF independent of these higher level differences.

IETF defines its XML syntax for geographical information as a subset of presence information (<presence> object in the XML example below), generally related to a device (<device> object) or a user (<user> object). IETF location information is represented using a Presence Information Data Format Location Object (PIDF-LO). This is represented as the <geopriv> object in the XML example below.

III.3.1 Example: Manual, GPS, AGPS, and External:CWMP <rootObject>.Location.{i}.DataObject. Format

This example, modified from an example in RFC5491, explains how to format location information in a <rootObject>.Location.{i}.DataObject. parameter with both geographical coordinates and civic location information according to the above-referenced IETF RFCs. The schema associated with the civic location namespace "urn:ietf:params:xml:ns:pidf:geopriv10:civicAddr" is specified in RFC5139 [9].

```
<presence xmlns="urn:ietf:params:xml:ns:pidf"
xmlns:dm="urn:ietf:params:xml:ns:pidf:data-model"
xmlns:gp="urn:ietf:params:xml:ns:pidf:geopriv10"
xmlns:gml="http://www.opengis.net/gml"
xmlns:cl="urn:ietf:params:xml:ns:pidf:geopriv10:civicAddr"
entity=" ">
      <dm:device id=" FFFFFF-FAP-123456789 ">
            <gp:geopriv>
                  <gp:location-info>
                        <gml:Point srsName="urn:ogc:def:crs:EPSG::4326">
                              <gml:pos>-43.5723 153.21760</gml:pos>
                        </gml:Point>
                        <cl:civicAddress>
                              <cl:FLR>2</cl:FLR>
```



```
                        </cl:civicAddress>
                    </gp:location-info>
                    <gp:usage-rules/>
                    <gp:method>Wiremap</gp:method>
                </gp:geopriv>
                <dm:deviceID>mac:8asd7d7d70</dm:deviceID>
                <dm:timestamp>2007-06-22T20:57:29Z</dm:timestamp>
        </dm:device>
</presence>
```

III.3.2 RFC5491 and RFC5139 Location Element Definitions

The XML elements are defined as follows by the IETF in RFC5491 [8] and related documents:

1. <presence> (RFC5491 [8])
   The <presence> element MUST have an 'entity' attribute. The value of the 'entity' attribute is the 'pres' URL of the presentity publishing this presence document.
   The<presence> element MUST contain a namespace declaration ('xmlns') to indicate the namespace on which the presence document is based. The presence document compliant to this specification MUST have the namespace 'urn:ietf:params:xml:ns:pidf:'. It MAY contain other namespace declarations for the extensions used in the presence XML document.
2. <device> (RFC5491 [8])
   The <device> element […] can appear as a child to <presence>. There can be zero or more occurrences of this element per document. Each <device> element has a mandatory "id" attribute, which contains the occurrence identifier for the device. In the TR-069 framework the id attribute will contain the CWMP Identifier of the device, in the form OUI-ProductClass-SerialNumber.
3. <geopriv> (RFC5491 [8], RFC5139 [9])
   Location information in a PIDF-LO can be described in a geospatial manner based on a subset of Geography Markup Language (GML) 3.1.1 or as civic location information specified in RFC5139[9]. The PIDF-LO Geodetic Shapes specification provides a specific GML profile for expressing commonly used shapes using simple GML representations. This profile defines eight shape types, the simplest ones being a 2-D and a 3-D Point. The PIDF-LO Geodetic Shapes specification also mandates the use of the World Geodetic System 1984 (WGS84) coordinate reference system and the usage of European Petroleum Survey Group (EPSG) code 4326 (as identified by the URN urn:ogc:def:crs:EPSG::4326) for two-dimensional (2d) shape representations and EPSG 4979 (as identified by the URN urn:ogc:def:crs:EPSG::4979) for three-dimensional (3d) volume representations.
   Each <geopriv> element must contain at least the following two child elements: <location-info> element and <usage-rules> element. One or more elements containing location information are contained inside a <location-info> element.
   a. <location-info> element can contain one or more elements bearing location information.
      i. <Point> element contains geographical data in the coordinate system specified by its srsName attribute. In the example above (WGS84/EPSG 4326), the syntax is latitude, longitude expressed in degrees

ii. Civic information elements are specified by IETF and can be added to the geographical data, though mixing information is not recommended.
iii. <relative-location> element is being proposed by IETF

b. <usage-rules> can contain the following optional elements:
i. <retransmission-allowed>: When the value of this element is 'no', the recipient of this Location Object is not permitted to share the enclosed Location Information, or the object as a whole, with other parties. RFC4119 [10] specifies that "by default, the value MUST be assumed to be 'no'".
ii. <retention expires>: This field specifies an absolute date at which time the Recipient is no longer permitted to possess the location information
iii. <external ruleset>: This field contains a URI that indicates where a fuller ruleset of policies, related to this object, can be found
iv. <notewell>: This field contains a block of text containing further generic privacy directives.

c. <method> is an optional element that describes the way that the location information was derived or discovered. Values allowed by RFC4119 [10] are stored in the IANA registry as "Method Tokens" [12]. The "Wiremap" value listed in the example is described as "Location determined using wiremap correlations to circuit identifiers "

4. <deviceID> element is mandatory. It contains a globally unique identifier, in the form of a URN, for each of the presentity devices (RFC4479[13])
5. <timestamp> is optional (RFC4479[13])

III.3.3 Use of RFC5491 and RFC5139 Location XML Elements in TR-069

1. <presence>
The entity attribute conveys no useful information and its value should be conventionally set to an empty string.
2. <device>
In RFC5491[8]this is one of the devices associated to the presentity. Devices are identified in the presence document by means of an instance identifier specified in the id attribute.
3. <geopriv>

a. <location-info>

2-D geographical coordinates with no additional civic information are sufficient in the simplest case.

i. <Point>

For 2-D applications the value of the srsName attribute should be set to the specified value "urn:ogc:def:crs:EPSG::4326"

b. <usage-rules>

ii. <retransmission-allowed>
Note that this field is not intended as instruction to the device whose location this is. Rather, it is intended to provide instruction to other systems that the device sends its location to (via SIP or other mechanisms). Therefore, the device will need to maintain its own policy (no standardized TR-069 data model is provided for this) as to when and where to send its location to others, and how to set this parameter when transmitting this location information. The device can choose to set this parameter to "yes" or to "no"

when sending its location to others. RFC4119[10] specifies that this element's default value is "no".

c. <method> If this location object is being created by the device as a result of GPS, AGPS, or Manual mechanisms, the <method> parameter will be populated with "GPS", "A-GPS", or "Manual", respectively. If the location object is being created by External:CWMP, then this parameter will not be used or populated by the ACS.

4. <deviceID> It contains a globally unique identifier, in the form of a URN, for each of the presentity devices (RFC4479 [13]).
5. <timestamp> is optional. The device (GPS, AGPS, Manual) or ACS (External:CWMP) can set this to the time the location was set or acquired.

Appendix IV. Fault Management

This section discusses the Theory of Operation for Fault Management using TR-069 [1] and the FaultMgmt object defined in the Root data model.

IV.1 Overview

There are four types of alarm event handling:

Expedited Event	Alarm event is immediately notified to the ACS with the use of Active Notification mechanism
Queued Event	Alarm event is notified to the ACS at the next opportunity with the use of Passive Notification mechanism
Logged Event	The CPE stores the alarm event locally but does not notify the ACS
Disabled Event	The CPE ignores the alarm event and takes no action

Note that all Fault Management tables are cleared when the device reboots.

Table 4 shows the multi-instance objects for FM to manage the alarm events.

Table 4 – FM Object Definition

Object name (<rootobject>.*Fault Mgmt.*)	Table size	Content	Purpose and usage
SupportedAlarm.{i}.	Fixed	Static & fixed content	Defines all alarms that the CPE supports. *ReportedMechanism* defines how the alarm is to be handled within the CPE: *0 – Expedited, 1 – Queued, 2 – Logged, 3 – Disabled* The table size is fixed and its content is static in order to drive the alarm handling behavior in the CPE.
ExpeditedEvent.{i}.	Fixed	Dynamically updated	Contains all *"Expedited"* type alarm events since the last device initialization. This includes events that are already reported or not yet reported to the ACS. One entry exists for each event. In other words, raising and clearing of the same alarm are two

Object name (<rootobject>.*Fault Mgmt.*)	Table size	Content	Purpose and usage
			separate entries. As the table size is fixed (vendor defined), new alarm event overwrites the oldest entry in FIFO fashion after the table becomes full.
QueuedEvent.{i}.	Fixed	Dynamically updated	Contains all *"Queued"* type alarm events since the last device initialization. This includes events that are already reported or not yet reported to the ACS. One entry exists for each event. In other words, raising and clearing of the same alarm are two separate entries. As the table size is fixed (vendor defined), new alarm event overwrites the oldest entry in FIFO fashion after the table becomes full.
CurrentAlarm.{i}.	Variable	Dynamically updated	Contains all the currently active alarms (i.e. outstanding alarms that are not yet cleared) since the last device initialization. When an outstanding alarm is cleared, that entry is deleted from this table. Therefore, only 1 entry exists for a given unique alarm. ACS can retrieve the content of this table to get the entire view of the currently outstanding alarms. As this is a variable size table, the size changes as alarm event is raised and cleared. If maximum entries for this table are reached, the next event overrides the object with instance number 1. Subsequent entries override objects at sequentially increasing instance numbers. This logic provides for automatic "rolling" of records. When a new alarm replaces an existing alarm, then all parameter values for that instance are considered as changed for the purposes of value change notifications to the ACS (even if their new values are identical to those of the prior alarm).
HistoryEvent.{i}.	Fixed	Dynamically updated	Contains all alarm events as a historical record keeping purpose. One entry exists

Object name (<rootobject>.*Fault Mgmt.*)	Table size	Content	Purpose and usage
			for each event. In other words, raising and clearing of the same alarm are two separate entries. The ACS can retrieve the content of this table to get the entire chronological history of the alarm events on the CPE. As the table size is fixed (vendor defined), new alarm event overwrites the oldest entry in FIFO fashion after the table becomes full.

Table 5 shows the timing of when an entry is to be created/updated/deleted.

Table 5 – FM Object Usage

Object name (<rootobject>.*FaultMgmt.*)	Timing of a new entry to be created	Timing of an existing entry to be updated	Timing of an existing entry to be deleted
ExpeditedEvent.{i}.	When a new event of *"Expedited"* type occurs (i.e. raise a new alarm or clear an existing alarm)	Never (i.e. once an entry is made, the content is not changed) The only exception is that when the table is rolling over in a FIFO fashion, the entry will be over-written.	Never (i.e. once created, the content is never deleted)
QueuedEvent.{i}.	When a new event of *"Queued"* type occurs (i.e. raise a new alarm or clear an existing alarm)	Never (i.e. once an entry is made, the content is not changed) The only exception is that when the table is rolling over in a FIFO fashion, the entry will be over-written.	Never (i.e. once created, the content is never deleted)
CurrentAlarm.{i}.	When a new alarm (all types except Disabled events) is raised	When the alarm status changes	When the alarm is cleared
HistoryEvent.{i}.	When a new event of all types except Disabled type occur (i.e. raise a new alarm or clear an existing alarm)	Never (i.e. once an entry is made, the content is not changed) The only exception is that when the table is rolling over in a FIFO fashion, the entry will be over-written.	Never (i.e. once created, the content is never deleted)

IV.1.1 Expedited Event

Figure 7 shows the expedited event handling. All alarms in the *"expedited"* type are stored in <rootobject>.*FaultMgmt.ExpeditedEvent.{i}.* multi-instance object and notified to the ACS using Active Notification mechanism by immediately establishing a TR-069 session with the ACS.

Alarms are also stored in <rootobject>.*FaultMgmt.CurrentAlarm.{i}.* and <rootobject>.*FaultMgmt.HistoryEvent.{i}.*

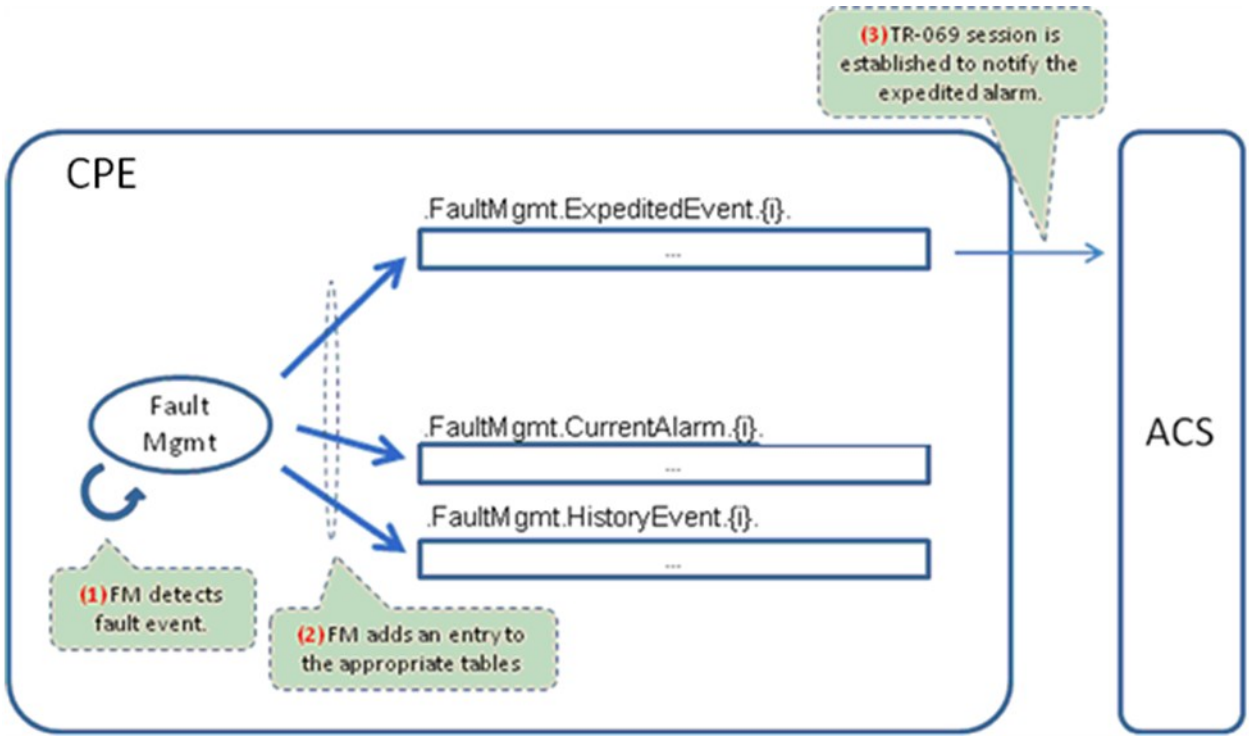

Figure 7 – Expedited Event Handling

IV.1.2 Queued Event

Figure 8 shows the queue event handling. All alarms in the "queued" type are stored in. <rootobject>.*FaultMgmtQueuedEvent*.{i}. multi-instance object. It is notified to the ACS using Passive Notification mechanism. In this case, the event is notified to the ACS at the next TR-069 session establishment.

Alarms are also stored in <rootobject>.*FaultMgmt.CurrentAlarm.{i}.* and <rootobject>.*FaultMgmt.HistoryEvent.{i}.*

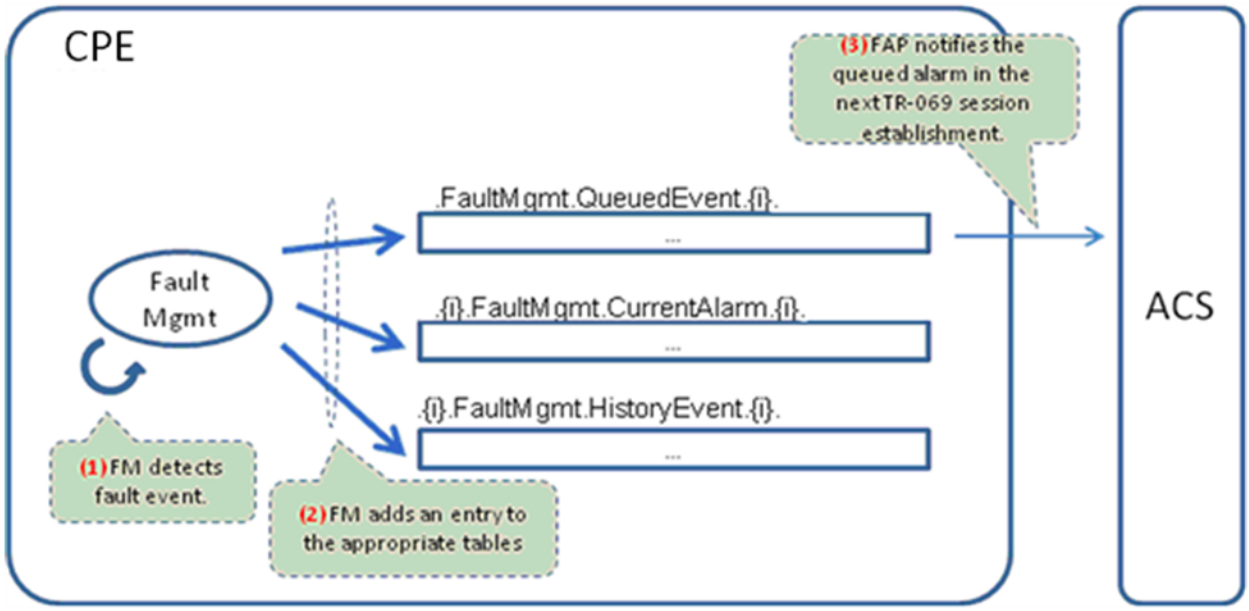

Figure 8 – Queued Event Handling

IV.1.3 Logged Event

Figure 9 shows the logged event handling. All alarms in the *"logged"* type are stored only in the <rootobject>.*FaultMgmt.CurrentAlarm.{i}.* and <rootobject>.*FaultMgmt.HistoryEvent.{i}.* Alarms of this type are not reported to the ACS.

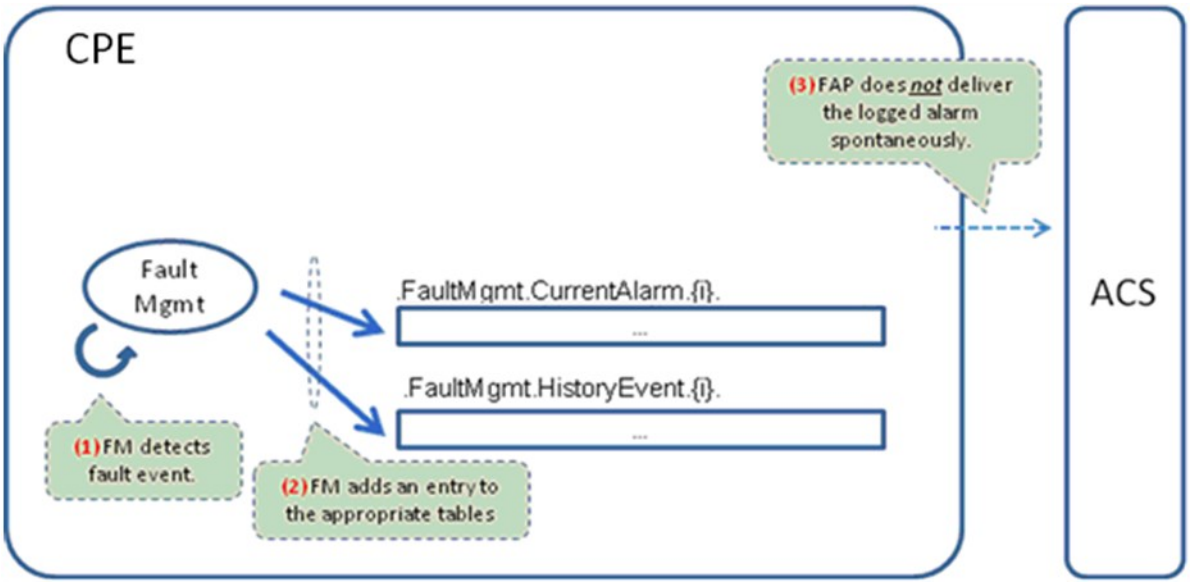

Figure 9 – Logged Event Handling

End of Broadband Forum Technical Report TR-157

