TECHNICAL REPORT

TR-330
TR-069 UPnP DM Proxy Management Guidelines

Issue: 1
Issue Date: August 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
1	24 August 2014	11 September 2015	Yu Zhu, Huawei Technologies Tim Spets, GreenWave Systems Wenchao Liu, China Telecom	Original

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

Editor	Tim Spets	GreenWave Systems
	Yu Zhu	Huawei Technologies
	Wenchao Liu	China Telecom
BroadbandHome™ WG Chairs	Jason Walls	QA Cafe
	John Blackford	Pace

Table of Contents

List of Figures

List of Tables

Executive Summary

This document provides a detailed implementation guide for proxying the UPnP Device Management (DM) Protocol [4] [5] [6] [7] with the TR-069 proxy management architecture defined in Annex J and Appendix I of TR-069 Amendment 5 [1] initially via the Virtual CWMP Device Mechanism and via the Embedded Object Mechanism in a later version of the document.

1 Purpose and Scope

1.1 Purpose

This document gives detailed implementation guidelines for using a CWMP enabled CPE Proxier to manage non-CWMP enabled devices that support the UPnP DM protocol.

1.2 Scope

This document complies with the proxy management mechanism and high level requirements specified in TR-069 Amendment 5 [1]. It further provides detailed solutions for implementing proxy management using TR-069 with UPnP DM.

This document provides guidelines for a UPnP DM protocol proxy solution on a CPE Proxier, and does not suggest modifying the UPnP DM protocol.

This document provides a solution that automates security setup for proxy management of UPnP Manageable [4] devices, and provides a solution for the proxier to authenticate the proxied devices including UPnP DM devices.

This document provides criteria for the Virtual CWMP Device proxy management mechanism, as it pertains to UPnP DM devices, and a future version will include the Embedded Object Mechanism for proxy management of UPnP DM devices.

1.3 References

The following references are of relevance to 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*	BBF	2013
[2]	RFC 2119	*Key words for use in RFCs to Indicate Requirement Levels*	IETF	1997
[3]	UPnP Device Architecture 2.0	*UPnP Device Architecture 2.0*	UPnP Forum	2012

[4]	UPnP Manageable Device	*UPnP Manageable Device v2*	UPnP Forum	2012
[5]	BMS	*UPnP Basic Management Service v2*	UPnP Forum	2012
[6]	CMS	*UPnP Configuration Management Service v2*	UPnP Forum	2012
[7]	SMS	*UPnP Software Management Service v2*	UPnP Forum	2012
[8]	DPS	*UPnP Device Protection Service v1*	UPnP Forum	2011

1.4 Definitions

The following terminology is used throughout this Technical Report.

Common Objects	A set of data model parameters defined by *UPnP DM* that models general information of a device.
CPE Proxier	A CPE that is capable of proxying operations between an ACS and a non-CWMP enabled device (i.e. a *Proxied Device*). There are two strategies for proxy management: *Virtual CWMP Device Mechanism* and *Embedded Object Mechanism*.
CWMP Endpoint	A CWMP termination point used by a CPE for Session communication with the ACS. This term is used interchangeably with CPE unless specifically defining behavior where a CPE supports multiple *CWMP Endpoints*.
Data Model	A hierarchical set of Parameters that define the managed Objects accessible via TR-069 for a particular device or service.
Embedded Object Mechanism	A proxy management strategy where the *CPE Proxier* embeds the details of the *Proxied Device* within the *Data Model*. The *Proxied Device* will appear to be integrated into the *CPE Proxier*.
Proxied Device	An endpoint that communicates indirectly with an ACS via a *CPE Proxier*.
UPnP Root Device	A logical device that is not embedded in any other logical device as specified in UPnP Device Architecture [3].
UPnP Embedded Device	A logical device that is embedded in a UPnP Root Device, as specified in UPnP Device Architecture [3].
UPnP DM	Term to reference the UPnP Device Management specifications.
UPnP DM Control Point	A UPnP Control Point that interacts with UPnP devices supporting at least one of the UPnP Device Management services.
UPnP DM Device	A *UPnP Root Device* or a *UPnP Embedded Device* that supports at least one of the *UPnP Device Management* services: the *Basic Management Service* [5], the *Configuration Management Service* [6], and the *Software Management Service* [7].

UPnP Manageable Device	A UPnP device type that utilizes *UPnP DM* functions by supporting *UPnP DM* services and *Common Objects*, as defined in *UPnP Manageable Device* [4].
UPnP Eventing	A behavior of a UPnP service to send notification of one or more changes in state variables to the Control Points that are subscribed to the service's events.
Virtual CWMP Device	A *CWMP Endpoint* virtualized by the *CPE Proxier* using the *Virtual CWMP Device Mechanism* to represent a *Proxied Device*.
Virtual CWMP Device Mechanism	A proxy management strategy where the *CPE Proxier* creates a virtual CWMP environment for the *Proxied Device*. The *CPE Proxier* provides a separate *CWMP Endpoint* for each such *Proxied Device*, which will therefore appear and be managed like a standalone CWMP enabled CPE.

1.5 Abbreviations

This Technical Report uses the following abbreviations:

ACS	Auto-Configuration Server
BMS	BasicManagement Service
CMS	ConfigurationManagement Service
CP	Control Point
CPE	Customer Premises Equipment
CWMP	CPE WAN Management Protocol
DCP	Device Control Protocol
DDD	Device Description Document
DHCP	Dynamic Host Configuration Protocol
DLNA	Digital Living Network Alliance
DM	Device Management
DPS	Device Protection Service
GENA	General Event Notification Architecture
OUI	Organizationally Unique Identifier
PC	Product Class
SCPD	Service Control Protocol Description
SMS	SoftwareManagement Service
SN	Serial Number
SSDP	Simple Service Discovery Protocol
TLS	Transport Layer Security
UDA	UPnP Device Architecture
URN	Uniform Resource Name

UUID	Universally Unique Identifier
WAN	Wide Area Network
WPS	Wi-Fi Protected Setup

2 Technical Report Impact

2.1 Energy Efficiency

This document has no impact on energy efficiency.

2.2 IPv6

This document has no impact on IPv6. UPnP Device Architecture (UDA) v2 [3] Annex A defines detailed IPv6 requirements for UPnP devices and Control Points. In general, UDAv2 requires all devices and Control Points to support dual stack (IPv4 and IPv6) operation, and operates on IPv4-only and IPv6-only networks. In this document, a proxied UPnP Device Management (DM) device that supports UDAv2 is expected to support IPv6, and to be able to communicate with the Control Point function of a CPE Proxier via both IPv4 and IPv6 interfaces.

2.3 Security

Before UPnP Device Protection Service (DPS) [8] was released, UPnP technology was often regarded as not secure. There were no security techniques, such as authentication, authorization and encryption, to control access to a UPnP device from different Control Points, or to provide confidentiality to the communications between them. Although there was a security specification called UPnP Device Security, it was not widely used due to its complexity. UPnP DPS has added security features to UPnP devices, and all UPnP specifications developed after that time have added support for DPS. This document is based on UPnP DMv2, which added support for DPS (section 4.8 provides more details). Based on UPnP DMv2 with DPS, the communication between the Control Point function of a CPE Proxier and a Proxied Device is secured via Transport Layer Security (TLS), and untrusted Control Points will be denied access to certain functions provided by the UPnP device.

2.4 Privacy

Devices supporting UPnP technology are usually consumer devices (e.g. audio/visual players, home automation devices etc.), which are not provisioned by telecom service providers. However, proxy management as discussed in this document, requires access to the UPnP DM device's information and functions from an ACS in the telecom network via a CPE Proxier (e.g. home gateway) within the consumer's home. This will incur privacy concerns, especially if proxy management is automatically enabled without a user's acknowledgement. To alleviate such privacy concerns, implementation of UPnP DM devices could display a notification to the user when a Control Point is requesting information or sending control messages, at least for the first time. When acknowledged by the user, proxy management can be enabled since the Control Point (residing on the CPE Proxier) is granted rights to access the UPnP DM device. Implementations could add this process at the introduction phase of the UPnP DPS procedure, as described in section 4.8.1.2.

3 Proxy Management Overview for UPnP DM

UPnP devices are commonly seen in the consumers' home network. These devices provide capabilities that range from user media sharing to remote printing and remote user control. In order to provide management features (e.g., troubleshooting, configuration, and software management) for users to manage their UPnP devices in the LAN, the UPnP Forum developed the UPnP DM Device Control Protocol (DCP) as defined in UPnP Manageable Device v2 [4]. Specifications for the UPnP DM contain one device specification ManageableDevice (MD) and three service specifications: BasicManagement Service (BMS), ConfigurationManagement Service (CMS), and SoftwareManagement Service (SMS). Together these service specifications provide troubleshooting diagnostics, configuration and software management.

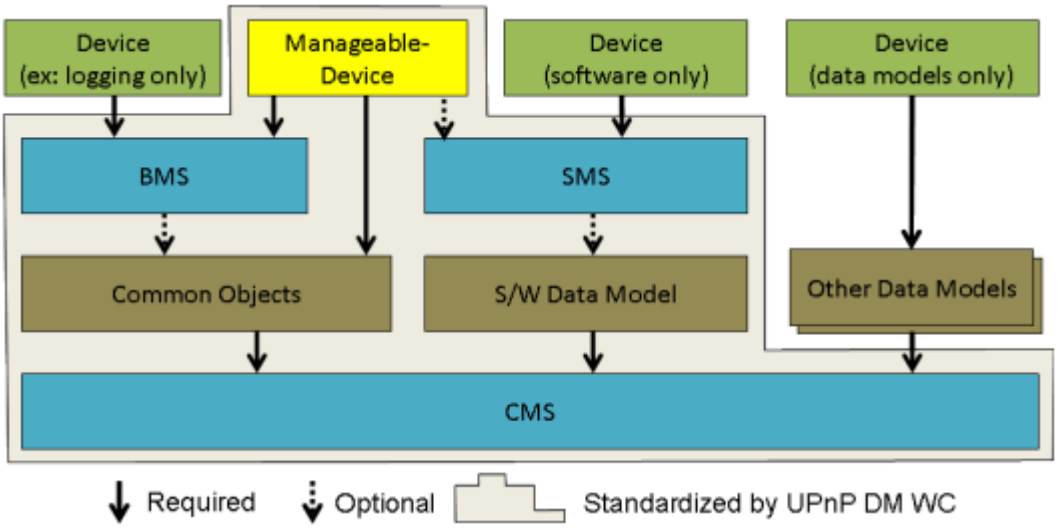

Figure 1 – UPnP Device Management Architecture Diagram

3.1 Proxy Management Architecture

In order to utilize the TR-069/UPnP DM proxy management solution as discussed in this document the following components of the solution require specific functionality as follows:

- The CPE Proxier will be compliant with Annex J of TR-069 Amendment 5 [1] (CWMP Proxy Management), and follow the guidelines in Appendix I of TR-069 Amendment 5[1] (CPE Proxier Implementation Guidelines).
- The UPnP DM devices will support at least one of the following UPnP DM v2 services: BMS, CMS or SMS. Support for the UPnP ManageableDevice (MD) as a container for these services is not required.

The CPE Proxier conceptually consists of three logical modules: CWMP client, TR-069/UPnP DM Proxy Module, and UPnP DM Control Point. CWMP requests received by the CWMP client from the ACS are translated by the TR-069/UPnP DM Proxy Module to the UPnP DM actions, and then passed to the UPnP DM Control Point to be sent to the UPnP DM devices. When a UPnP action response or event is received by the UPnP DM Control Point, the action response and event is passed to the TR-069/UPnP DM Proxy Module to be converted to a CWMP response or CWMP event to be sent to the ACS.

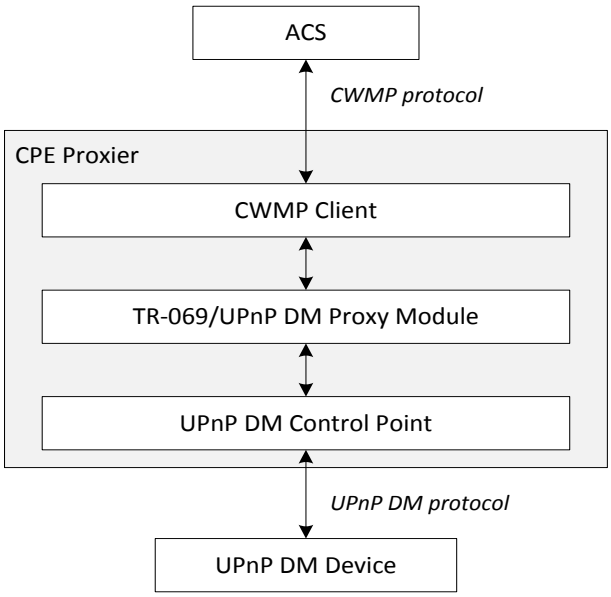

Figure 2 – TR-069/UPnP DM Proxy Management Architecture

TR-069 [1] provides the Virtual CWMP Device Mechanism and Embedded Object Mechanism to correlate device application functionality with proxy modules. Architectures that utilize these mechanisms are provided in following sections. Implementations can use one or both of the mechanisms based on specific deployment scenarios.

3.1.1 Proxy Management Architecture for Virtual CWMP Device Mechanism

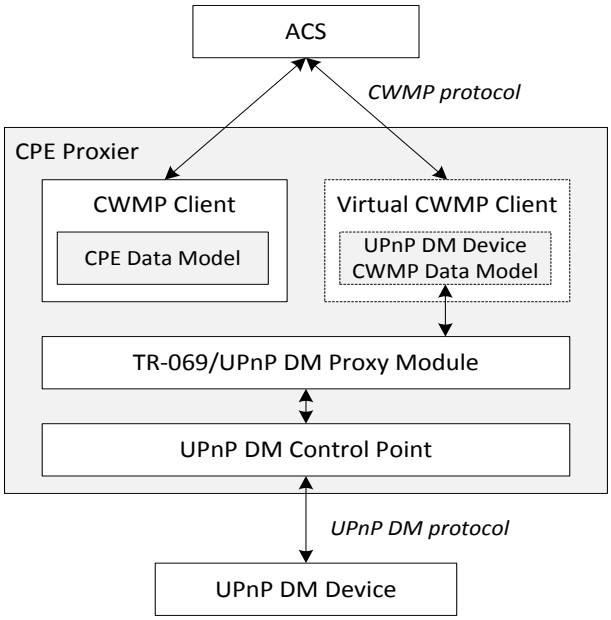

Figure 3 – TR-069/UPnP DM Proxy Management Architecture for Virtual CWMP Device Mechanism

The architecture includes three entities:

1) The ACS, which communicates with the CPE Proxier using the CWMP protocol.
2) The CPE Proxier, serving as a management proxy between the ACS and the UPnP DM device, uses the following components:
   - A CWMP client with CPE data models.
   - One or more Virtual CWMP Devices each containing TR-069 formatted Device data models that are proxied using UPnP DM.
   - A TR-069/UPnP DM Proxy Module that translates between UPnP DM actions/events and CWMP requests and responses, and interacts with the Virtual CWMP Device(s) and the UPnP DM Control Point using internal interfaces.
   - A UPnP DM Control Point that communicates with the UPnP DM device(s) using the UPnP DM protocol.
3) One or more UPnP DM devices that support at least one of the UPnP DM services (i.e. BMS, CMS, and SMS).

3.1.2 Proxy Management Architecture for Embedded Object Mechanism

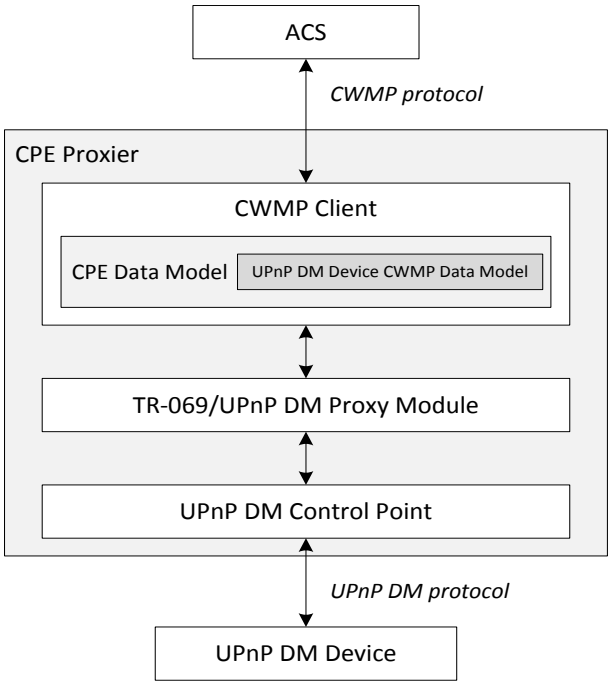

Figure 4 – TR-069/UPnP DM Proxy Management Architecture for Embedded Object Mechanism

The architecture includes three entities:

1) The ACS, which communicates with the CPE Proxier using CWMP protocol.
2) The CPE Proxier, severing as a management proxy between the ACS and the UPnP DM device, uses the following components:
   - A CWMP client with CPE data models. The CPE's data model includes application objects that through which the ACS interacts; data model for embedded devices and for the UPnP DM proxy and discovery protocols.
   - A TR-069/UPnP DM Proxy Module that translates between UPnP DM actions/events and CWMP request and response, and interacts with the CWMP client and the UPnP DM Control Point using internal interfaces.
   - A UPnP DM Control Point that communicates with the UPnP DM device using the UPnP DM protocol.
3) One or more UPnP DM devices that support at least one of the UPnP DM services (i.e. BMS, CMS, and SMS).

Note that the proxy management architecture for UPnP DM devices using the Embedded Object Mechanism is only for information in this version of the guidelines.

3.2 TR-069/UPnP DM Proxy Module Functionalities

The TR-069/UPnP DM Proxy Module performs the protocol translation operations and is considered to be the main component of this document. In general, the proxy module has following functionalities or responsibilities:

1) Construct the data models for Virtual CWMP Device or Embedded Object after UPnP DM devices are discovered by the UPnP DM Control Point.
2) Translate the CWMP requests received from the CWMP Client to UPnP DM actions, and pass them to the UPnP DM Control Point. In turn, translate the action responses received from the UPnP DM Control Point to CWMP responses and pass them to the CWMP Client.
3) Convert the UPnP events, when received from the UPnP DM Control Point, into data model parameter value changes.

4 Proxy Management Implementation Guidelines for UPnP DM

4.1 UPnP Overview

UPnP defines devices using a flexible model described in UPnP ManageableDevice [4]. A UPnP device is a logical device that supports one or more UPnP services. The UPnP devices that are discussed in this document support at least one UPnP DM service in order to be proxy managed and are referred to as UPnP DM devices.

The UPnP Forum has defined a device type known as a 'ManageableDevice'. A UPnP ManageableDevice may be a UPnP Root or UPnP Embedded Device. A UPnP ManageableDevice must support UPnP BasicManagement Service (BMS), UPnP ConfigurationManagement Service (CMS) including the UPnP DM Common Objects (required in order to provide support for OUI/PC/SN) and may support UPnP SoftwareManagement Service (SMS).

UPnP DM services are designed to be included in other types of UPnP devices besides ManageableDevice, as discussed in the UPnP ManageableDevice [4]. They include examples of UPnP Root and UPnP Embedded Devices using a mixture of UPnP BMS, CMS and SMS services.

A physical device may contain multiple UPnP devices modeled using a UPnP ManageableDevice and UPnP Device (implementing UPnP DM services) both modeled as UPnP Root devices. A physical device may contain a single UPnP Root device with a UPnP ManageableDevice as a UPnP Embedded Device (implementing UPnP DM services).

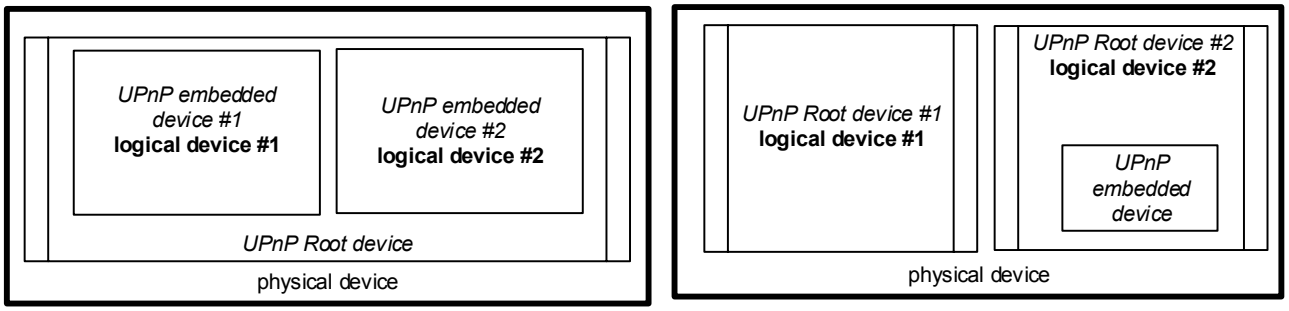

Figure 5 – Physical devices and UPnP logical devices represented with UPnP Root and UPnP Embedded Devices

4.1.1 Guidelines for Discovering UPnP Devices

Any UPnP device implementing one or more of the UPnP DM services may be considered as a manageable UPnP device and may be managed by the ACS through CPE Proxier.

Due to UPnP, advertising of the device may be infrequent or disabled, and the queries and advertisements are sent with a UDP transport (unreliable). The CPE Proxier performs queries for supported UPnP DM devices on regular (possibly configurable) intervals.

When discovering UPnP DM devices the CPE Proxier takes these steps:
1) Receive an advertisement for a supported UPnP DM service.
2) Retrieve the root DDD from the UPnP DM service advertisement.
3) Verify if the reported UPnP DM service is supported by the CPE Proxier.
4) Retrieve the SCPD for supported UPnP DM service (CMS, BMS, and SMS).

UPnP Root or UPnP Embedded Devices may be supported, and may be considered a unique proxy entity, i.e. a unique Virtual CWMP Device or Embedded Object instance in the CPE Proxier's data model.

Virtual CWMP Device Requirements [1]:

- The CPE Proxier MUST obtain a unique OUI and Serial Number from the Proxied Device.
- The Proxied Device MUST support a Reboot Mechanism.

The CPE Proxier queries for Common Objects (one method is the UPnP CMS::GetSupportedDataModels() action). If UPnP CMS and Common Objects are supported, the Device uses the OUI/PC/SN support.

In some cases UPnP devices may use the same UUID or Common Objects over multiple devices on the same physical device. Implementations will model these UPnP devices as a single CWMP Device.

In the case where OUI/PC/SN are not available, each UPnP device may advertise with the same UUID and different MAC address will be modeled as a different Virtual CWMP Device. This may result in Multi-Home UPnP devices appearing twice. This may be resolved if the UPnP device utilizes the same IP Port number (Location field) for each device.

When a Proxied Device goes offline or has been rediscovered, follow TR-069a5 [1] J.2.3. Utilize any of the available UPnP CMS/BMS service queries to determine if the Proxied Device is online or offline.

4.2 UPnP Eventing and CWMP Inform Events

UPnP Eventing Mechanism from UDA 2.0 [3]
The below *excerpts* detail critical operations and mechanisms in UPnP eventing referenced in the following sections.

"After a control point has (1) discovered a device and (2) retrieved a description of the device and its services, the control point has the essentials for eventing." "If one or more of these state variables are evented, then the service publishes updates when these variables change, and a control point may ***subscribe*** *to receive this information."*

- *"To subscribe to UPnP Eventing, a subscriber sends a subscription message. If the subscription is accepted, the publisher responds with a duration for the subscription. To*

*keep the subscription active, a subscriber must **renew** its subscription before the subscription expires."*

- *"The publisher notes changes to state variables by sending event messages. Event messages contain the names of one of more state variables and the current value of those variables, expressed in XML. A special initial event message is sent when a subscriber first subscribes; this event message contains the names and values for all evented variables and allows the subscriber to initialize its model of the state of the service."*

UPnP uses Announcements to Advertise) conditions such as:

- *ssdp:update (Device Update) sent for each of the root devices (immediate), embedded devices and embedded services. This advertises a new interface when enabled or lost/reconnected.*
- *ssdp:alive (Notify) , When a device is added to the network it MUST advertise its root devices (immediate), embedded devices and embedded services.*
- *ssdp:byebye (Device Unavailable), This SHOULD be sent when a Device and its services are going to be removed from the network.*
- *Search – sent by Control Point to solicit unicast Advertisements*.
- *UPnP DM describes a new Announcement called Announcement.dm.upnp.org (UPnP DM BMS section 2.4.1) "The mechanism MUST only be used to announce important Parent Device state information that cannot reasonably be sent using GENA eventing."*

4.2.1 Guidelines for Mapping UPnP Events and Announcements to CWMP Inform Events

For a CPE Proxier to perform proxy management for a UPnP DM Device utilizing the UPnP Eventing Mechanism, the CPE Proxier maintains an active subscription to the supported UPnP services (SMS/BMS/CMS).

Detailed operations and use of Events are detailed in the related sections.

The following table shows the high level mapping between the CWMP Inform Events and UPnP DM Events / SSDP messages. CWMP Inform Events, which are not in the table (i.e. “2 Periodic” and “3 Scheduled” etc.), are expected to be handled by the CPE Proxier itself and have no link to the UPnP DM events.

Table 1 – High Level Mapping Between CWMP Events and UPnP DM Events

CWMP Inform Event	Associated UPnP DM Event / State Variable and Announcement Message	Guidelines for CPE Proxier
0 BOOTSTRAP	Announcements: - ssdp:byebye - Announcement.dm.upnp.org : AboutToBaselineReset, - ssdp:alive If a UPnP device receives a UPnP BMS::BaselineReset() request, an AboutToBaselineReset Announcement is sent.	When the CPE Proxier Discovers the UPnP DM device for the first time by receiving a ssdp:alive (reference section 4.1.1) it sends a "0 BOOTSTRAP" event, as discussed in TR-069a5 [1] Appendix I.4.1 bullet #2. When the CPE Proxier receives ssdp:byebye containing the Announcement.dm.upnp.org: AboutToBaselineReset header of a UPnP DM Device. The CPE Proxier will cache this notification for use after receiving a later ssdp:alive message from the same proxied UPnP device. Then the CPE Proxier sends a "0 BOOTSTRAP" to the ACS. If the ACS URL of a Proxied Device has been modified, the CPE Proxier sends a "0 BOOTSTRAP" event to the ACS.

CWMP Inform Event	Associated UPnP DM Event / State Variable and Announcement Message	Guidelines for CPE Proxier
1 BOOT (M Reboot)	Announcements: - ssdp:byebye/ - Announcement.dm.upnp.org : AboutToReboot, - ssdp:alive If the UPnP BMS::BaselineReset() request involves a reboot, the ssdp:byebye message will also include Announcement.dm.upnp.org: AboutToReboot header.	When the CPE Proxier receives ssdp:byebye from the Proxied UPnP device with Announcement.dm.upnp.org: AboutToReboot header (this may be the result of the Proxied UPnP device accepting a UPnP BMS::Reboot() action from the CPE Proxier), the CPE Proxier should cache this notification for use. Then, when receiving ssdp:alive message from the same Proxied UPnP Device, the CPE Proxier sends "1 BOOT" event to the ACS. If the UPnP BMS::Reboot() invocation is originated from the ACS via the CWMP Reboot RPC method, then a "M Reboot" is sent by the CPE Proxier.
4 VALUE CHANGE	UPnP CMS Event: - ConfigurationUpdate A ConfigurationUpdate event is the result of at least one parameter who's EventOnChange attribute is True and has changed.	When the CPE Proxier receives the UPnP CMS Event with ConfigurationUpdate, the CPE Proxier queries all parameters with EventOnChange attribute set (excluding those with a recent modification from the ACS) and compare to a cached value to determine which have changed (reference section 4.6.5.2.1). The CPE Proxier sends "4 VALUE CHANGE" event to the ACS, and include the changed parameters in the INFORM Parameter List.
7 TRANSFER COMPLETE M Download / M ScheduleDownload M Upload	SMS Event: - OperationIDs [Update()] OperationID is assigned and added to the OperationIDs when the UPnP DM SMS action (only applied for DU which DUID is 0, firmware) is successfully requested. OperationID is used in calling SMS Operations: GetOperationInfo(), GetOperationIDs(), The OperationID is removed from the OperationIDs when the software management operation completes. In the UPnP SMS Event the OperationIDs will be sent. GetOperationIDs is used to return a list of OperationIDs that are currently active.	The CPE Proxier determines which related event(s) to send to the ACS. When one of the following operations were initiated by the CPE Proxier has been finished, the CPE Proxier receives an SMS Event with OperationIDs and it verifies the associated OperationID is no longer active, or may poll using the UPnP SMS::GetOperationIDs() action: For CWMP Download and ScheduleDownload RPCs (UPnP SMS::Update() action), and Upload Logfiles (UPnP BMS::GetLogInfo() action) send CWMP Inform Event with "7 TRANSFER COMPLETE" ("M Download", "M ScheduleDownload", "M Upload") see sections 4.4 and 4.7.1. With the "7 TRANSFER COMPLETE" event, the CPE Proxier also sends the CWMP TransferComplete RPC to the ACS. The calling arguments of the TransferComplete are mapped (see sections 4.4 and 4.7.1) from the outputs of the associated UPnP actions and RPCs. Upload or Download of Configuration files is not supported in UPnP DM.

CWMP Inform Event	Associated UPnP DM Event / State Variable and Announcement Message	Guidelines for CPE Proxier
8 DIAGNOSTICS COMPLETE	UPnP BMS Event: - ActiveTestIDs A TestID is added to the ActiveTestIDs when the test is successfully requested. The TestID is removed from the ActiveTestIDs when the test completes. In the UPnP BMS Event the ActiveTestIDs will be sent. GetActiveTestIDs is used to return a list of TestIDs that are currently active.	When a test that was initiated by the CPE Proxier has been finished, the CPE Proxier receives a UPnP BMS Event with ActiveTestIDs, and verifies the associated TestID is no longer active, or may poll using the UPnP BMS::GetActiveTestIDs() action, the CPE Proxier sends "8 DIAGNOSTICS COMPLETE" event to the ACS (reference section 4.5).
10 AUTONOMOUS TRANSFER COMPLETE	SMS Event: - OperationIDs [Update()] Same description as in the row of “7 TRANSFER COMPLETE” above.	The CPE Proxier should track the SMS::OperationIDs event in order to determine a firmware update happened. When the CPE Proxier receives the SMS::OperationIDs event (when a DU operation is not requested via the ACS), it calls the UPnP SMS::GetOperationInfo() action for each OperationID from the OperationIDs list. If the value of the returned Action argument is “Update”, and the value of the returned TargetIDs argument is 0 for the first ID, this indicates a firmware update operation was requested by another Control Point. The CPE Proxier should cache this OperationID, and verify if it is removed from a later SMS::OperationIDs event. If so, the CPE Proxier sends the “10 AUTONOMOUS TRANSFER COMPLETE" CWMP Inform Event to the ACS. Note that this CWMP Inform Event will not be sent when the autonomous transfer complete policy is set to disabled in the Virtual CWMP Device’s data model. Reference section 4.7.1.3 for more details.

4.3 UPnP Device Data Model

In UPnP DM the data model and data structure are similar to CWMP data models (hierarchical, name defined, array objects etc.). Device vendors are also allowed to define additional data models based on the UPnP DM criteria.

The CPE Proxier will utilize the defined Device Type Schema based on CWMP Data Model Schemas and Vendor extensions it supports with potentially multiple Device.DeviceInfo.SupportedDataModel.{i}.URL for each Virtual CWMP Device type it supports.

4.3.1 UPnP Device Data Model Proxy Guidelines

When the UPnP device is discovered with UPnP CMS service, the CPE Proxier decides if the SupportedDataModel returned from a UPnP CMS::GetSupportedDataModels() action matches (at least a subset of) a Device.DeviceInfo.SupportedDataModel.{i} instance described above for the Virtual CWMP Device.

The CPE Proxier instantiates the VirtualDevice table for the Virtual CWMP Device and increments the value of the ManagementServer.VirtualDeviceNumberOfEntries parameter, in its Device:2 Data Model. If notification is set on the parameter, the CPE Proxier notifies the ACS using the “4 VALUE CHANGE” event.

Additionally, the CPE Proxier verifies that all of the required UPnP CMS actions for proxy support are in the SCPD. If there are any writeable parameters, then the UPnP CMS::SetValues() action needs to be supported. If there are any writable Multi-Instance objects, then the UPnP CMS::AddInstance(), and the CMS::DeleteInstance() actions need to be supported.

4.3.2 UPnP Services Supported by CWMP Object Model

There are objects and parameters that exist in the CWMP Device:2 object model that may be accessed via UPnP services and may not exist in the UPnP DM data model (Ping, NSLookup etc.). Therefore they will not appear in the parameter hierarchy on the UPnP DM device when queried.

4.3.2.1 Services Support in Object Model Guidelines

To build the objects to support UPnP BMS or SMS actions, the CPE Proxier discovers which BMS or SMS actions are supported, and create the appropriate parameters and objects for each UPnP action that is supported.

For example, if UPnP BMS::Ping() is supported (and BMS::GetPingResults(), BMS::GetActiveTestIDs(), BMS::GetTestInfo()), then there will be a Device:2 IPPing object to support it created. Same goes for other BMS diagnostics / log management, and SMS actions for Execution Units etc.

The CPE Proxier holds the noted objects and parameters locally and inserts them when queries such as CWMP GetParameterNames and CWMP GetParameterValues RPCs are requested where they are in the ParameterPath.

4.4 UPnP Logging

UPnP BMS defines GetLogURIs(), SetLogInfo() and GetLogInfo() actions to manage logs. In CWMP, an ACS can read Device.DeviceInfo.VendorLogFile.{i}. object of the Device:2 data model for basic information of vendor logs, and the ACS can send a CWMP Upload RPC method (FileType = "4 Vendor Log File <i>") to retrieve a log file identified by its instance number.

4.4.1 Mapping UPnP Log Info to CWMP VendorLogFile Guidelines

As a startup process, a CPE Proxier updates its VendorLogFile object of the Device:2 data model to reflect the current logs supported by the UPnP device. The CPE Proxier can perform the following steps.

- Call UPnP BMS::GetLogURIs() to retrieve a list of URIs (LogURIs) of the logs currently supported.
- Call UPnP BMS::GetLogInfo() for each LogURI to retrieve information of each log.
- Create a Device:2 VendorLogFile instance for each of the logs, and map internally the instance number with the BMS LogURI, and then initialize the parameter values:

When the ACS reads the log instances by calling the GetParameterNames or GetParameterValues CWMP RPC methods, the CPE Proxier communicates with the UPnP device by calling the UPnP BMS::GetLogURIs() and the UPnP BMS::GetLogInfo() actions, and then refreshes its data model if needed before responding, since BMS allows logs to be added and removed at run time. The value of the Device:2 VendorLogFile object parameters can be decided as follows.

- Name: allocate and use an internal file system path name.
- MaximumSize: use the value of LogMaxSize returned by UPnP BMS::GetLogInfo().
- Persistent: could use 0 initially, and observe if the log content is purged after the device rebooted and update the value accordingly, since BMS has not defined an argument to indicate log persistency.

4.4.2 Uploading Log File Guidelines

When requested by ACS to upload a particular log file (identified by instance number) using the CWMP Upload RPC method, the CPE Proxier downloads the log file from the UPnP device and then uploads it to the ACS.

- Verify RPC Method is correct, if not return error 9011 and exit.
- If verified, accept the RPC method and return Status = 1.
- Call the UPnP BMS::GetLogInfo() action with the LogURI that the instance number mapped with.
- Download the log file from the UPnP device by the LogURL the action returned. (CPE Proxier could choose to begin uploading when the log file is completely or partially downloaded from the UPnP device.
- Upload the log file to the location as the ACS requested.
- Initiate a CWMP session with the "7 TRANSFER COMPLETE" event and the "M Upload" event, and then send the CWMP TransferComplete RPC method. If failed when downloading the log file from the UPnP device, use 9011 as fault code value in the CWMP TransferComplete RPC.

4.5 UPnP Diagnostics

UPnP performs diagnostics via services (BMS). CWMP performs diagnostics via the supported Device:2 Data Model.

4.5.1 Diagnostics Guidelines

In order to perform the diagnostics function the UPnP Control Point subscribes to UPnP BMS events (DeviceStatus, SequenceMode, TestIDs, and ActiveTestIDs). For UPnP Eventing details please reference section 4.2.

Figure 6 provides the logic to perform when a CPE Proxier receives a diagnostic request via the CWMP SetParameterValues RPC method to the diagnostic object in the Data Model, and executes an associate UPnP DM Service.

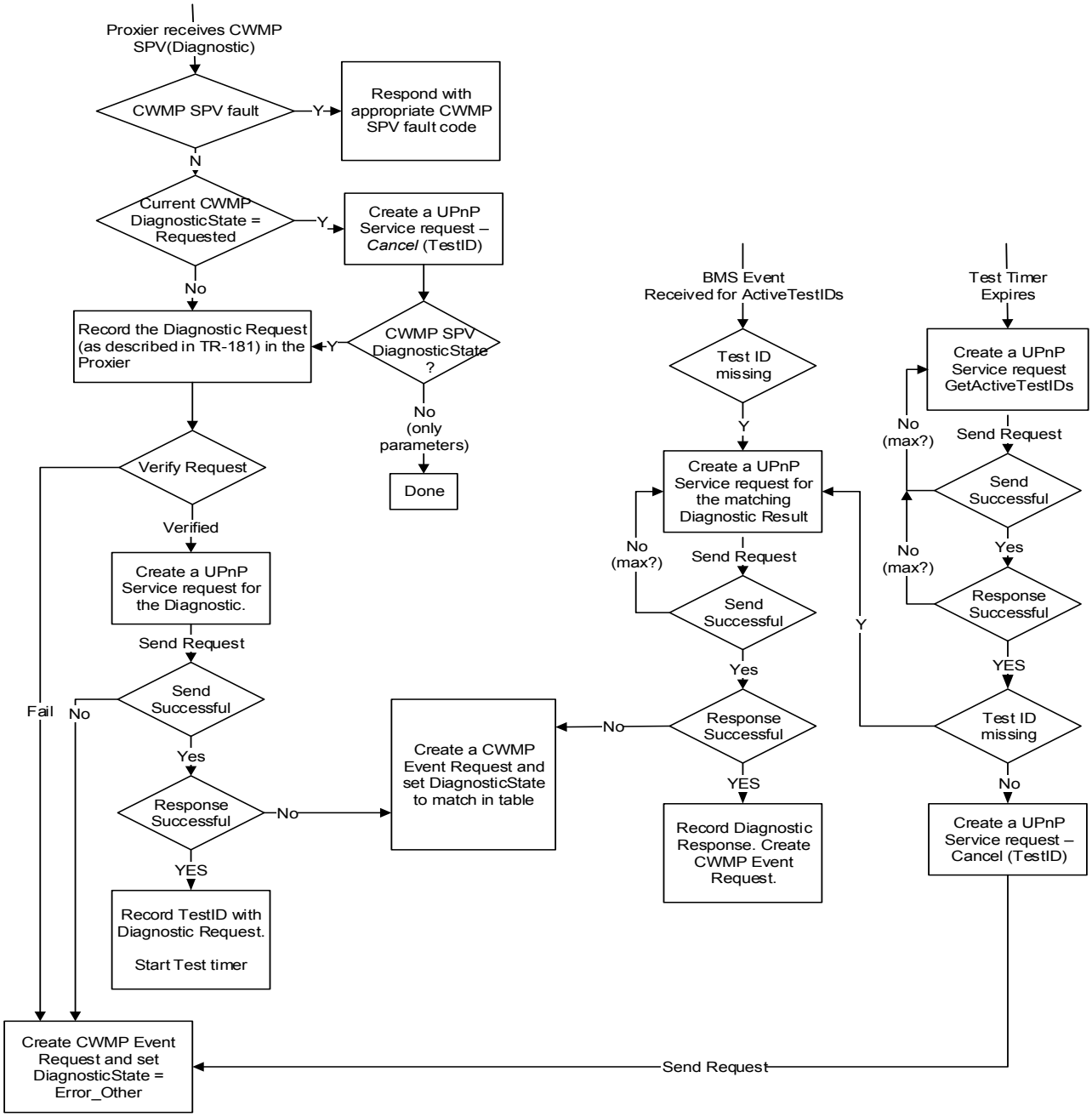

Figure 6 – Multi-Threaded Diagnostic Guideline Logic

Notes:

- Where “max ?” is noted there will likely be a limit to the retries, if this is hit then Create Event Request and set DiagnosticState = Error_Other.
- The Test timer is based on the type of test, number of repetitions, test parameters etc.
- Create Event request for “8 Diagnostics Complete”.
- GetTestIDs was not chosen since if the TestID is present a Diagnostic Results request may fail for 703 (Test active), which would require a call to cancel the test.
- The nature of the Diagnostic tests does not require a connection to the Device to respond to the test request (it is suggested the test is not executed until the session completes). For this reason the only failure of a Diagnostic within the session of the request is a SetParametersValue failure.
- Given that the Diagnostic result is recorded in the CPE Proxier, a link availability test is not required prior to starting the session to send the CWMP “8 DIAGNOSTICS COMPLETE” Inform Event. The ACS can retrieve the test results from the CPE Proxier cache without requiring a connection to the CPE.

4.6 Configuration and Data Model Management

UPnP CMS commands more closely resemble CWMP RPCs and the similarities/differences/issues are documented below. In some cases UPnP CMS offers more functionality for similar functions. In this case the additionally functionality is noted, but not relevant in the proxy operation. For the cases where CWMP offers more functionality, the guidelines below will give behavior for the functional area.

4.6.1 Configuration overview

Below is a comparison of CWMP data model syntax and UPnP CMS data model Syntax.

Note: UPnP data model instance numbers are 0 based, CWMP data model instance numbers are 1 based. In the example below the instance numbering is arbitrary.

CWMP Syntax:
ParameterPath = Device.Hosts.Host.1.IPAddress
UPnP Syntax:
/RootPath/SingleInstanceNodeName “Device”/ SingleInstanceNodeName “Hosts” /
MultiInstanceNodeName “Host”/ Instance “1”/ LeafName “IPAddress”

CWMP Syntax:
InstancePath = Device.Hosts.Host.1.
UPnP Syntax:
/RootPath/SingleInstanceNodeName “Device”/ SingleInstanceNodeName “Hosts” /
MultiInstanceNodeName “Host”/ Instance “1”/

CWMP Syntax:
MultiInstancePath = Device.Hosts.Host.
UPnP Syntax:
/RootPath/SingleInstanceNodeName “Device”/ SingleInstanceNodeName “Hosts” /
MultiInstanceNodeName “Host”/

CWMP Syntax:
SingleInstancePath = Device.Hosts.
UPnP Syntax:
/RootPath/SingleInstanceNodeName “Device”/ SingleInstanceNodeName “Hosts” /

4.6.2 CWMP GetParameterNames

The CWMP GetParameterNames RPC retrieves the current **instantiated** parameters, objects, object instances, parameters of all object instances and the write attribute of all returned (all depending on calling arguments). CWMP GetParameterNames has a NextLevel argument that dictates the amount of the data model to retrieve.

UPnP CMS::GetSupportedParameters() action supports a single instance ParameterPaths Starting Node and Search Depth. This will return all the Parameters in the StructurePaths to the Search Depth (without instance numbers).

UPnP CMS::GetInstances() action supports a PartialPath Starting Node and Search Depth. This will return a list of all possible PartialPaths with instance numbers.

UPnP CMS::GetAttributes() action supports a list of ParameterPaths (ending in parameters), MultiInstancePaths (ending in Multi-Instance object), InstancePaths (ending in instances) and attributes returned (type, access, EventOnChange) apply to Parameters and objects.

4.6.2.1 CWMP GetParameterNames Guidelines

All Parameters are returned along with a Writeable Boolean.

Guideline Steps

1) Call UPnP CMS::GetInstances (ParameterPath, depth=1), this will give the CPE Proxier all the instances of the object.
2) Call UPnP CMS::GetSupportedParameters (ParameterPath, depth=1). This will give the CPE Proxier all the parameters of the object.
3) Call UPnP CMS::GetAttributes() action with a ParameterPath
    - For each single instance object use the original ParameterPath/ Each Parameter from UPnP CMS::GetSupportedParameters() action in step 2 as input argument.
    - For Multi-Instance objects use each ParameterPath/ output of UPnP CMS::GetInstances() action in step one and one for / Each Parameter from UPnP CMS::GetSupportedParameters() action in step 2 as input argument.

4) If successful, format the CWMP GetParameterNames response with the full parameter path and Writeable = True if UPnP CMS::Access attribute returns readWrite, else Writeable = False.
5) If UPnP CMS::GetAttributes() action returns a 703 error code, then return CWMP GetParameterNamesResponse with a 9005 error code.

Note: An implementation could take an approach where the entire object model of the proxied device is held in the CPE Proxier, in this case the Writeable attribute is known a-priori, and doing a UPnP CMS::GetAttributes() for the parameter is not necessary.

Guideline considerations:

- If the ParameterPath is a parameter – Go to step #3
- If the NextLevel is true and ParameterPath is a single instance object – Go to step #2
- If the NextLevel is true and ParameterPath is a multiple instances object – Go to step #1, skip step #2.
- If the NextLevel is false and the ParameterPath is an object – Start at Step #1 use node depths = 0 in steps 1 & 2.

Guideline Examples:

Example of where ParameterPath is a parameter:
CWMP GetParameterNames (Device.Hosts.Host.1.IPAddress, true)
UPnP CMS::GetAttributes (Device/Hosts/Host/1/IPAddress, 1)

Example of where ParameterPath is a single instance object node:
CWMP GetParameterNames (Device.Hosts., true)
UPnP CMS::GetSupportedParameters (Device/Hosts/, 1)
Returns Device/Hosts/Host
Returns Device/Hosts/HostNumberofEntries
UPnP CMS::GetAttributes (Device/Hosts/Host/)
UPnP CMS::GetAttributes (Device/Hosts/HostNumberofEntries)

Example of where ParameterPath is a multiple instance object Nextlevel = true:
CWMP GetParameterNames (Device.1.Hosts/Host., true)
UPnP CMS::GetInstances (Device/Hosts/Host, 1)
Returns Device/Hosts/Host/1/
Returns Device/Hosts/Host/3/
UPnP CMS::GetAttributes (Device/Hosts/Host/1/)
UPnP CMS::GetAttributes (Device/Hosts/Host/3/)

Example of where ParameterPath is an object and NextLevel is false:
CWMP GetParameterNames (Device.Hosts., False)
UPnP CMS::GetInstances (Device/Hosts/Host, 0)
Returns Device/Hosts/Host/1/
Returns Device/Hosts/Host/3/
UPnP CMS::GetSupportedParameters (Device/Hosts/, 0)
Returns Device/Hosts

Returns Device/Hosts/HostNumberofEntries
Returns Device/Hosts/Host/
Returns Device/Hosts/Host/#/<parameter 1>
:
Returns Device/Hosts/Host/#/<parameter x>
UPnP CMS::GetAttributes (Device/Hosts/Host/)
UPnP CMS::GetAttributes (Device/Hosts/HostNumberofEntries)
UPnP CMS::GetAttributes (Device/Hosts/Host/1/)
UPnP CMS::GetAttributes (Device/Hosts/Host/1/<parameter 1>)
:
UPnP CMS::GetAttributes (Device/Hosts/Host/1/<parameter x>)
UPnP CMS::GetAttributes (Device/Hosts/Host/3/)
UPnP CMS::GetAttributes (Device/Hosts/Host/3/<parameter 1>)
:
UPnP CMS::GetAttributes (Device/Hosts/Host/3/<parameter x>)

Guideline Exceptions: Including non-Data Model elements

As described in 4.3.2 there are Data Model objects in CWMP that are represented as services in UPnP DM. In this case, those service elements are 'inserted' at the appropriate objects in the Data Model. An example of this is a CWMP GetParameterNames on the root node of the UPnP DM device with next level=false. This is described in Figure 7.

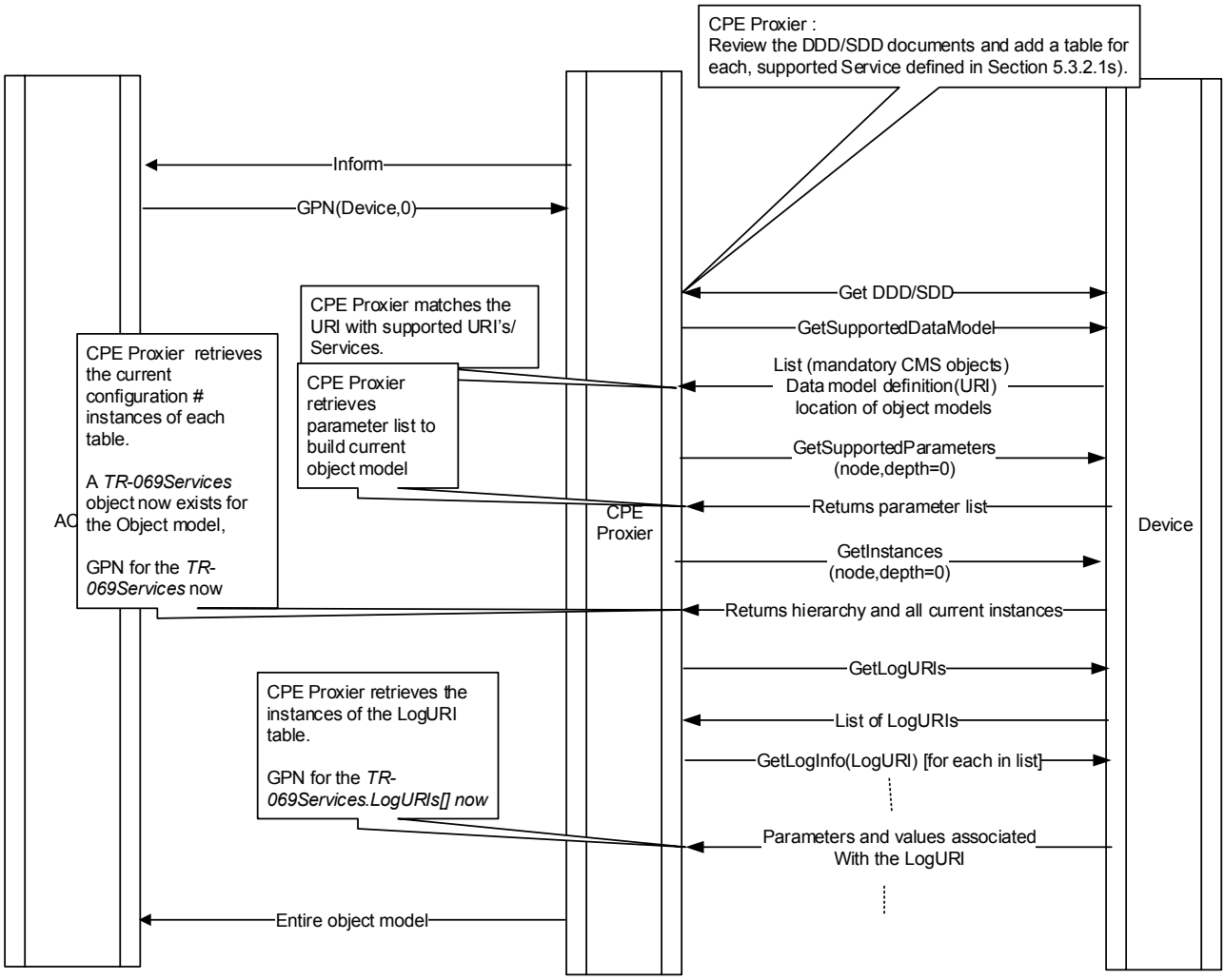

Figure 7 – CWMP GetParameterNames Root Device, next level = false

4.6.3 CWMP GetParameterValues

The CWMP GetParameterValues retrieves the current value(s) of any parameter or list of parameters returns the parameter(s) and their associated value(s). CWMP GetParameterValues of a partial path (object) requests parameter values for all parameters in the branch of the hierarchy.

UPnP CMS::GetValues() accepts list of 'Content Paths' which may contain ParameterPath (ending in a parameter), a SingleInstancePath (single instance object), MultiInstancePath (multiple instance object/table), or an InstancePath (specific instance of a multiple instance object). If a Content Path is given then all descendant parameters and values are returned.

The CWMP GetParameterValues RPC and the UPnP CMS::GetValues() action operate identically for single parameters, list of parameters, and all Partial paths.

4.6.3.1 CWMP GetParameterValues Guidelines

Although the commands are a direct mapping, the Proxy needs to fill in the 'xsi:type' in the CWMP GetParameterValues response acquired from the CWMP Data Model file.

If the UPnP CMS::GetValues() action returns an error code, the associated GetParameterValuesResponse error code can be found in Table 2.

4.6.4 CWMP SetParameterValues

The CWMP SetParameterValues and UPnP CMS::SetValues() both require the device to apply the changes to all of the specified parameters atomically, or they return a committed response. Both commands allow a list of parameter-value pairs as input arguments.

The CWMP SetParameterValues and UPnP CMS::SetValues() both allow a response of committed and applied. UPnP CMS::SetValues() response of committed is in line with the CWMP SetParameterValues language in [1]. UPnP CMS::SetValues() response of committed insures that subsequent reading of the parameters will reflect the new values.

The UPnP CMS::SetValues() command returns values of either committed or applied. If the UPnP DM device returns a committed response it applies the changes as soon as possible.

4.6.4.1 CWMP SetParameterValues Guideline

The commands are able to support a direct mapping, The CPE Proxier relays the UPnP CMS::SetValues() response to the ACS, i.e.:

- If the value of the Status output argument of UPnP CMS::SetValues() action is "ChangesCommitted", the CPE Proxier will use "Status=1" in the CWMP SetParameterValuesResponse.
- If the value of the Status output argument of UPnP CMS::SetValues() action is "ChangesApplied", the CPE will use "Status=0" in the CWMP SetParameterValuesResponse.

Note that all UPnP CMS "write" actions for Data Model, i.e. SetValues(), SetAttributes(), CreateInstance() and DeleteInstance() return the same "Status" parameter. The CPE Proxier guidelines for handling the corresponding CWMP RPC response (if supports Status) are the same as described above.

If UPnP CMS::SetValues() returns an error code, the associated CWMP SetParameterValuesResponse error code can be found in Table 2.

Note: When processing a CWMP SetParameterValues command with multiple parameters, if there is a failure CWMP requires a fault code for each parameter that failed. In this case the request should return a failure without details.

4.6.5 CWMP Attributes and UPnP DM CMS Attributes

The UPnP CMS attributes (described in [6] 2.3.2) are persistent across discovery. This section discusses the UPnP CMS attributes used by the CWMP proxy.

The UPnP CMS Access attributes values are either RO (Read Only) or RW (Read Write) and apply to parameters, Multi-Instance objects, and instance objects.

When access applies to a table, [MultinistancePath] if set to RW then rows [Instance Nodes] can be created using UPnP CMS::CreateInstance() action. When access applies to a table row, [InstancePath] if set to RW then the particular row can be deleted using UPnP CMS::DeleteInstance() action. When access applies to a parameter [Leaf Node], if it is set to RW then value can be modified via a UPnP CMS::SetValues() action.

The UPnP CMS EventOnChange attributes (RW) [boolean] applies to parameters and Multi-Instance objects.

When the Control Point subscribes to UPnP DM Events for CMS::ConfigurationUpdate, all parameters and tables with the EventOnChange set to 1 will generate an update to the CMS::ConfigurationUpdate [associated UPnP CMS::GetConfigurationUpdate() action**]** state variable and generate an event.

The UPnP CMS::ConfigurationUpdate state variable will be updated [for subscribed nodes] if a parameter valued is updated or an instance is created/deleted from a table.

Parameters that support the UPnP CMS::EventOnChange attribute will be specified in the associated data model. EventOnChange will report an event when a change occurs to the parameter due to a UPnP action or some other event out of UPnP control.

4.6.5.1 CWMP GetParameterAttributes

The attributes returned from a CWMP GetParameterAttributes only apply to parameters, but may be requested on a partial path (single instance object, Multi-Instance object, object instance). For each parameter that is a partial path, the response will include ALL parameters in the branch of the naming hierarchy.

UPnP CMS::GetAttributes() action supports a list of ParameterPath (ending in a parameter), MultiInstancePath (ending in Multi-Instance object) or InstancePath (ending in instances), attributes apply to parameters and objects.

4.6.5.1.1 CWMP GetParameterAttributes Guidelines

For CWMP GetParameterAttributes on Parameters there is a direct mapping to UPnP CMS::GetAttributes(), but only the EventOnChange is mapped to the Notification. When returning a CWMP GetParameterAttributes response, AccessList will always be empty.

For CWMP GetParameterAttributes requested with a partial path, the CPE Proxier submits a separate UPnP CMS::GetAttributes() action for each parameter below that point in the naming hierarchy and uses the information to fill the ParameterList in the response.

Guideline Steps

1) Call UPnP CMS::GetInstances (ParameterPath, depth=0), this will give the CPE Proxier all the instances of the object.
2) Call UPnP CMS::GetSupportedParameters (ParameterPath, depth=0). This will give the CPE Proxier all the parameters of the object.
3) Call UPnP CMS::GetAttributes() action with a ParameterPath
    - For each single instance object use the original ParameterPath/ Each Parameter from UPnP CMS::GetSupportedParameters() action in step 2 as input argument.
    - For Multi-Instance objects use each ParameterPath/ output of UPnP CMS::GetInstances() action in step one and one for / Each Parameter from UPnP CMS::GetSupportedParameters() action in step 2 as input argument.

These steps are only performed on the Parameters (not objects or object instances).

If UPnP CMS::GetAttributes() action returns an error code, reference Table 2 for the CWMP GetParameterAttributesResponse error code.

Note: An implementation could take an approach where the entire object model of the proxied device is held in the CPE Proxier, in this case current attributes may already be known and doing a UPnP CMS::GetAttributes() for the parameter is not necessary.

4.6.5.2 CWMP SetParameterAttributes

CWMP SetParameterAttributes applies to parameters, but may be requested on a partial path (single instance object, Multi-Instance object, object instance) to apply the same changes to all parameters below that point in the hierarchy. Both CWMP SetParameterAttributes and UPnP CMS::SetAttributes() action is successful for all requested changes atomically, or this action will fail and all changes will be removed.

The UPnP CMS::SetAttributes() action returns values of committed or applied. If the device returns a committed response the device applies the changes as soon as possible.

4.6.5.2.1 CWMP SetParameterAttributes Guidelines

The CWMP SetParameterAttributes guidelines only apply to Notifications (see 4.2). The CPE Proxier sends a 9003 error response when the CWMP SetParameterAttributes request attempts to set the AccessList value**.** The UPnP CMS::SetAttributes() action is only sets the EventOnChange attribute.

Note: Reference "4 VALUE CHANGE" in the Event Handling section 4.2.1. When a Parameter has EventOnChange set to true the CPE Proxier keeps a cached copy of this setting for reference when a ConfigurationUpdate Event is received.

For CWMP SetParameterAttributes requested with a partial path, the CPE Proxier submits a UPnP CMS::SetAttributes() for each parameter below that point in the naming hierarchy. To do this the CPE Proxier performs the following steps:

Guideline Steps

1) Call UPnP CMS::GetInstances (ParameterPath, depth=0). This will give the CPE Proxier all the instances of the object.
2) Call UPnP CMS::GetSupportedParameters (ParameterPath, depth=0). This will give the CPE Proxier all the parameters of the object.
3) Call UPnP CMS::SetAttributes() action with a ParameterPath
    - For each single instance object use the original ParameterPath/ Each Parameter from UPnP CMS::GetSupportedParameters() in step 2 as input argument.
    - For Multi-Instance objects use each ParameterPath/ output of UPnP CMS::GetInstances() action in step one and one for / Each Parameter from UPnP CMS::GetSupportedParameters() action in step 2 as input argument.

If any UPnP CMS::SetAttributes() action fails, repeat the above by resetting to the Parameter attributes in the cache.

If UPnP CMS::SetAttributes() action returns an error code, the associated CWMP SetParameterAttributesResponse error code can be found in Table 2. If UPnP CMS::SetAttributes() action returns a committed response it is treated the same (see 4.6.4.1) as the applied response.

Note: An implementation could take an approach where the entire object model of the proxied device is held in the CPE Proxier, in this case current attributes may already be known and doing a UPnP CMS::GetAttributes() for the parameter is not necessary.

4.6.6 CWMP AddObject

CWMP AddObject relies on the defined object model to provide initialization values for the newly created parameters. The CWMP AddObject and UPnP CMS::CreateInstance() action both allow a response of committed and applied. The CWMP AddObject and UPnP CMS::CreateInstance() action both have the return value of the instance created.

UPnP CMS::CreateInstance() action arguments include a Multi-Instance path and Initialization values for the table entries created (done with a RelativePath and value pair). UPnP CMS::CreateInstance() action returns an InstanceIdentifier that is InstancePath from the Root Node to the Instance Node, whereas the CWMP AddObject only return the instance number.

UPnP CMS::CreateInstance() action response of committed is in line with the CWMP AddObject rules in TR-069 [1].

4.6.6.1 CWMP AddObject Guidelines

When the CPE Proxier receives a CWMP AddObject RPC it maps the UPnP CMS::CreateInstance() InstanceIdentifier in UPnP device to the InstanceIdentifier in the CPE Proxier.

The CPE Proxier may call UPnP CMS::CreateInstance() with the default values provided in the defined object model. This would require the CPE Proxier to contain the necessary object model parameter defaults for all possible tables.

The CPE Proxier relays the Status value returned by the UPnP CMS::CreateInstance() call to the Status input argument of the CWMP AddObjectResponse:

- If the value of the Status output argument of UPnP CMS::AddObject() action is "ChangesCommitted", the CPE Proxier uses "Status=1" in the CWMP AddObjectResponse.
- If the value of the Status output argument of UPnP CMS::AddObject() action is "ChangesApplied", the CPE uses "Status=0" in the CWMP AddObjectResponse.

If UPnP CMS::CreateInstance() action returns an error code, the associated CWMP AddObjectResponse error code can be found in Table 2.

4.6.7 CWMP DeleteObject

UPnP CMS::DeleteInstance() action deletes one instance. The CWMP DeleteObject and UPnP CMS::DeleteInstance() action both allow a response of committed and applied.

4.6.7.1 CWMP DeleteObject Guidelines

UPnP CMS::DeleteInstance() deletes the InstanceNode (object instance) giving in the calling argument. The CPE Proxier relays the Status value returned by the CMS::DeleteInstance() call to the Status input argument of the CWMP DeleteObjectResponse,.

- If the value of the Status output argument of UPnP CMS::DeleteInstance() action is "ChangesCommitted", the CPE Proxier uses "Status=1" in the CWMP DeleteObjectResponse.
- If the value of the Status output argument of UPnP CMS::DeleteInstance() action is "ChangesApplied", the CPE uses "Status=0" in the CWMP DeleteObjectResponse.

If UPnP CMS::DeleteInstance() action returns an error code, the associated CWMP DeleteObjectResponse error code can be found in Table 2.

4.6.8 CWMP Alias support

For CPE Proxiers that support the CWMP Instance Alias, the CPE Proxier retains a cached copy of all Instance Aliases and manages them according to CWMP rules (see TR-069 Amendment 5 [1] section 3.6.1).

4.6.9 UPnP CMS Error Code Mapping

Mapping of UPnP UDA 2.0 [3] error codes and CMS error codes (701-709) to the CWMP Fault Codes are described in Table 2.

Table 2 – UPnP Error Codes Mapping for CWMP Faults

UDA 2.0 /CMS ErrorCode	Error Description	CWMP Fault Code	Comments
401	Invalid Action	9001 Request denied	Note: For UPnP errors not specified or discussed in this document (e.g. 613-699, 800-899 etc.), CPE Proxier return 9001 to the ACS.
402	Invalid Args	9003 Invalid arguments	
403	*(Do Not Use)*	--	
501	Action Failed	9001 Request denied	
600	Argument Value Invalid	9003 Invalid arguments	
601	Argument Value Out of Range	9003 Invalid arguments	
602	Optional Action Not Implemented	9001 Request denied	
603	Out of Memory	9004 Resources exceeded	
604	Human Intervention Required	9001 Request denied	
606	Action not authorized	9002 Internal Error	
605	String Argument Too Long	9003 Invalid arguments	
606-612	*Reserved (for DeviceSecurity)*	9001 Request denied	
613-699	*TBD*	9001 Request denied	
701	Invalid Argument Syntax	9003 Invalid arguments	
702	Invalid XML Argument	9003 Invalid arguments	
703	No Such Name	9005 Invalid Parameter Name	
704	Invalid Value Type	9006 Invalid Parameter type	
705	Invalid Value	9007 Invalid Parameter Value	
706	Read Only Violation	9008 Attempt to set a non-writeable parameter	
708	Resource Temporarily Unavailable	9004 Resources exceeded	
709	Resources Exceeded	9004 Resources exceeded	
800-899	*TBD*	9001 Request denied	

4.7 Firmware and Software Module Management

UPnP SoftwareManagement Service (SMS) introduces actions for managing software installation on a UPnP DM device. The scope of SMS covers the full support of software module management and firmware management. SMS reserves DUID 0 and EUID 0 for the management of firmware. Firmware only performs the UPnP SMS::Update() action, using the DUID 0 as the identifier.

This version of the document only provides proxy management guidelines for firmware. A future version of this document may cover proxy management for UPnP SMS software modules which are not firmware.

4.7.1 Firmware Management Guidelines

The DUID 0 is firmware. A UPnP Control Point (CP) can send UPnP SMS::Update() action requested on DUID 0 to update the firmware of the device. When this is requested the UPnP device immediately returns an OperationID. The unique OperationID will be used to track the state of the operation (Requested, InProgress, Error, and Completed). The OperationID may be queried by using a UPnP SMS::GetOperationInfo() action at any time. The Control Point may also subscribe to the SMS::OperationIDs state variable to receive events when the OperationID state changes to either Completed or Error.

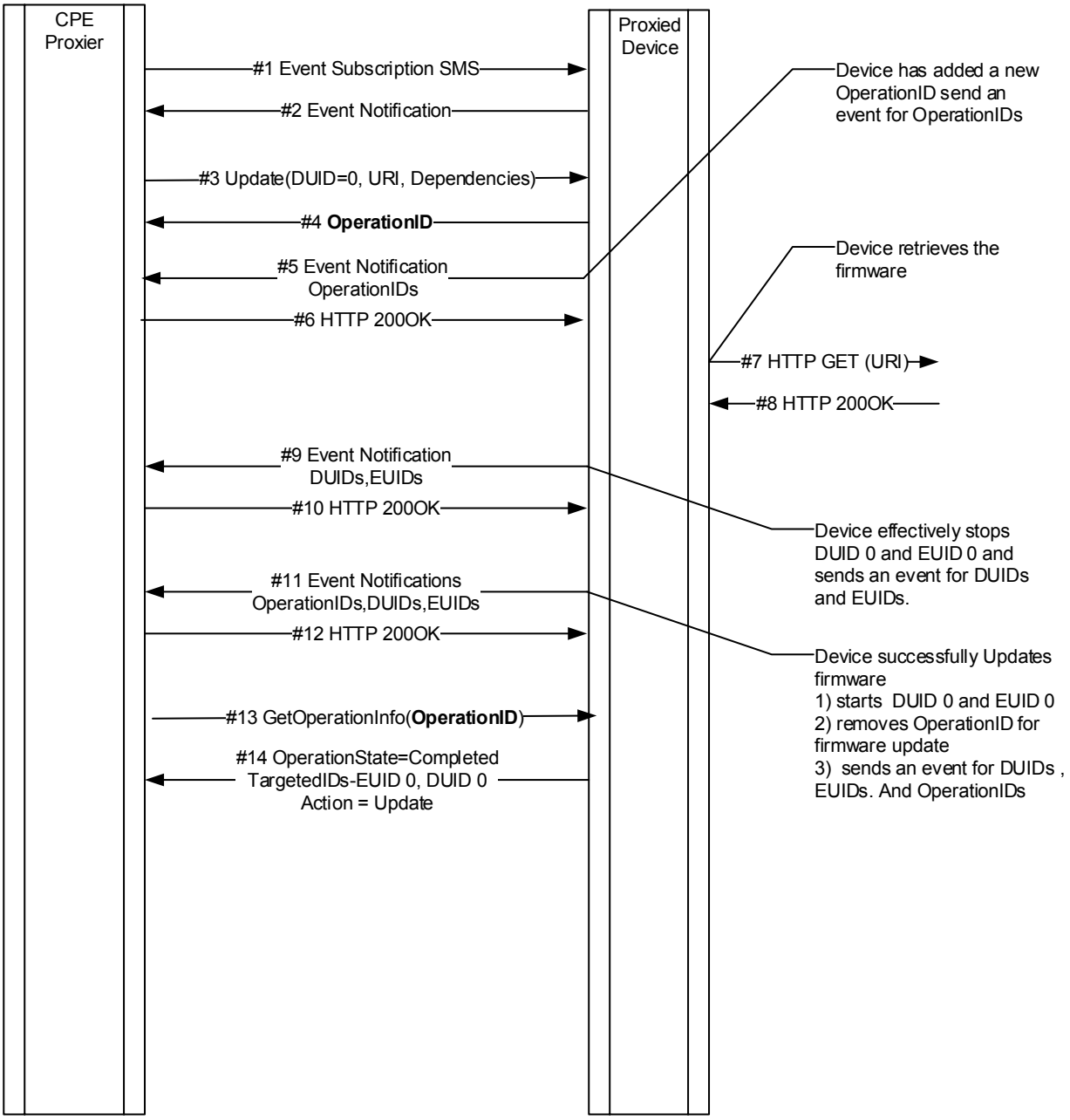

Figure 8 – Sequence of Events for successful Firmware update

Figure 8 represents a typical successful update scenario. The event subscription (#1) is not required, and the CPE Proxier may choose to poll the UPnP SMS::GetOperationInfo() (#13) messages.

Exception Handling:

1) UPnP SMS::GetOperationInfo() (#13) returns Error. This gives the CPE Proxier an indication that the SMS::Update() action was not successful. This could be the result of an unsuccessful firmware download (#7) or failed firmware verification checks after the firmware is downloaded (#8).

2) If the Proxied Device requires a Reboot (between #10 and #11) there is no requirement for the device to retain any OperationID information, this will result in step #11 never being received and any UPnP SMS::GetOperationInfo() (#13) performed after the reboot will likely return an error stating the OperationID does not exist.
    - The CPE Proxier might check the version value either from SMS::GetDUInfo() action or from the SMS Software Data Model to determine the update operation succeeds.
    - The Proxied Device may FAIL in its implementation of the new firmware and ROLLBACK to the previous version. The CPE Proxier will only receive the initial device advertisements described in 4.1.1.

4.7.1.1 CMWP Download Guidelines

An ACS can request a CPE to perform a firmware upgrade by sending the CWMP Download or ScheduleDownload RPC methods. The optional CWMP ScheduleDownload RPC method is not recommended to be implemented by a CPE Proxier, since the time window logic is neither supported in UPnP SMS nor easy to be performed by a CPE Proxier in proxy management.

The UPnP SMS::Update() action has DUID, NewDUURI and HandleDependencies input arguments, and returns OperationID. When receiving the CWMP Download RPC request with argument FileType="1 Firmware Upgrade Image", a CPE Proxier performs the following guideline steps:

1) The CPE Proxier first validates the RPC request.
2) If the validation is successful, the CPE Proxier returns Status=1 to the ACS.
3) The CPE Proxier calls the SMS::Update() action.
    - Using 0 for the value of the DUID input argument.
    - Using the value of the URL argument of the CWMP Download request as the NewDUURI input argument.
    - Using 0 for the value of the HandleDependencies input argument, to avoid potential UPnP SMS errors.
    - The CPE Proxier can observe the DelaySeconds argument of the CWMP Download by applying an equivalent delay before calling the UPnP SMS::Update() action.
4) If the SMS::Update() action is accepted by the Proxied Device, the CPE Proxier caches the returned OperationID, and associates it with the CommandKey (if not empty string) sent in the CWMP Download RPC request. The same CommandKey will be returned to the ACS via a subsequent CWMP TransferComplete RPC. The OperationID is used both to identify if an SMS Operation is requested by the CPE Proxier (via ACS) and to track the operation status.

Guidelines for result retrieval and ACS report are given in section 4.7.1.2, and guidelines for handling of errors are given in Table 3. Note that a subsequent CWMP TransferComplete RPC will deliver any errors that occur after the CWMP Download RPC request has been accepted by the CPE Proxier.

4.7.1.2 CMWP Inform Event and TransferComplete Guidelines

The UPnP SMS::OperationIDs event (or the SMS::GetOperationIDs() action) and the SMS::GetOperationInfo() action are means for a CPE Proxier to determine if a firmware update operation has successfully finished or failed. Below are guideline steps for the CPE Proxier to retrieve operation result from the Proxied Device, and then report to the ACS using the CWMP Inform RPC and TransferComplete RPC methods.

1) The CPE Proxier tracks the UPnP SMS::OperationIDs event (or by polling the SMS::GetOperationIDs() action), when the update operation completes the corresponding OperationID (which is cached by the CPE Proxier) will be removed from the list.
    - Note that there may be a condition that OperationIDs is dropped and no event sent because of a reboot after firmware upgrade. The CPE Proxier might check the version value either from UPnP SMS::GetDUInfo() action or from the SMS Software Data Model to determine if the update operation succeeded.
2) Then the CPE Proxier calls the SMS::GetOperationInfo() action with the cached OperationID to confirm the result of the update operation.
    - If the value of the returned ErrorDescription argument of the action is “Error_None”, it indicates the operation succeeded. Other values indicate operation failed.
3) The CPE Proxier initiates a CWMP session and delivers the “7 TRANSFER COMPLETE” event and the “M Download” event in the CWMP Inform RPC method.
4) In the same CWMP session the CPE Proxier then sends the CWMP TransferComplete RPC method, with the arguments handled as follows.
    - CommandKey: use the same CommandKey when the CPE Proxier received the CWMP Download RPC method for this firmware upgrade operation, which was associated with the SMS OperationID returned by the Proxied Device.
    - FaultStruct: mappings with the UPnP SMS ErrorDescription and AdditionalInfo are detailed in Table 3.
    - StartTime: the time when the OperationID from the corresponding UPnP SMS::Update() action (see Figure 8 flow #3) is added to the SMS::OperationIDs state variable (the CPE Proxier needs to record the event).
    - CompleteTime: the time when the associated OperationID is removed from the UPnP SMS::OperationIDs state variable (see Figure 8 flow #11, the CPE Proxier needs to record the event). If no events is received because the Proxied Device rebooted, the CPE Proxier can use the time when the SSDP:alive is received.

Table 3 – Error Case Mappings for Firmware Update Operation

UPnP DM Action	Error / Output Argument	Allowed Values	FaultStruct Value of TransferComplete	*Comments*
SMS::Update()	701 Invalid URI		9001 Request Denied	*Use 9001 since 9003 is not supported by the RPC.*
	702 HandleDependencies request not honored		9001 Request Denied	*This is not expected to happen if the CPE Proxier would not request handle dependencies.*
	704 Already in a transitory state		9001 Request Denied	
	705 Invalid DUID		9001 Request Denied	*This is not expected to happen for firmware (DUID=0 is always supported).*
	710 NewDUURI not supported		9001 Request Denied	
SMS::GetOperationInfo()	ErrorDescription	Error_None	FaultCode=0, empty FaultString	
		Error_ConcurrentAccess	FaultCode=9001	
		Error_MissingDependency	FaultCode=9001	
		Error_Network	FaultCode=9015	
		Error_CorruptedFile	FaultCode=9018	
		Error_DiskFull	FaultCode=9027	
		Error Other	FaultCode=9001	
	AdditionalInfo	*any string*	FaultString	*Use the value for FaultString when error occurs.*

4.7.1.3 CMWP Inform Event and AutonomousTransferComplete Guidelines

UPnP SMS allows different Control Points (CPs) to manage firmware or software modules of a UPnP DM device. In this case, a CPE Proxier might know when a firmware update is requested by another CP, and report the result to the ACS via the CWMP "10 AUTONOMOUS TRANSFER COMPLETE" Inform event and the CWMP AutonomousTransferComplete RPC method. Guideline steps are as follows.

1) The CPE Proxier tracks the UPnP SMS::OperationIDs event (or by polling the SMS::GetOperationIDs() action), to retrieve every OperationID for which the associated software operation has been requested but has not yet finished or failed.

2) Then the CPE Proxier calls the SMS::GetOperationInfo() for each OperationID that is not requested (previously cached) by the CPE Proxier via the ACS. The firmware update operation is determined if following conditions are all met.
    - The value of the returned Action argument is “Update”.
    - The first ID of the returned TargetedIDs argument is 0.
3) If an ongoing firmware update operation is identified, the CPE Proxier caches this specific OperationID. The CPE Proxier also records the time when this SMS::OperationIDs event is received, which will be used as the StartTime for the CWMP AutonomousTransferComplete RPC method.
4) Then the CPE Proxier listens for a future SMS::OperationIDs event, from which the cached OperationID is removed (indicating the operation succeeded or failed). The CPE Proxier also records the time of receiving this event, which will be used as CompleteTime of the CWMP AutonomousTransferComplete RPC method.
    - Note that there is a condition where an OperationID is dropped and no event is sent because of a reboot after firmware upgrade.
    - In this case, the CPE Proxier could check the version value either from UPnP SMS::GetDUInfo() action or from the SMS Software Data Model to determine if the update operation succeeded, and the CompleteTime would be the time that the UPnP SSDP:alive was received after the Proxied Device rebooted.
5) The CPE Proxier calls the SMS::GetOperationInfo() action with the cached OperationID. The operation result is determined from the ErrorDescription output argument, same as discussed in section 4.7.1.2.
6) Then the CPE Proxier initiates a CWMP session and delivers the “10 AUTONOMOUS TRANSFER COMPLETE” event in the CWMP Inform RPC method.
7) In the CWMP session, then the CPE Proxier sends the CWMP AutonomousTransferComplete RPC method, with the arguments handled as follows.
    - AnnounceURL: use empty string.
    - TransferURL: call the SMS::GetDUInfo() action with DUID=0, and use the returned DUURI as the value. Or, use the value of the /UPnP/DM/Software/DU/0/URI parameter in the SMS Software Data Model. If the Proxied Device supports neither interface, then an empty string could be used.
    - IsDownload: use 1 (true).
    - FileType: use "1 Firmware Upgrade Image".
    - FileSize: use zero, or if supported by the Proxied Device, use the value of the /UPnP/DM/Software/DU/0/Size parameter in the SMS Software Data Model.
    - TargetFileName: use empty string.
    - FaultStruct: same as discussed in section 4.7.1.2 (detailed in Table 3).
    - StartTime: use the time as determined in step 3).
    - CompleteTime: use the time as determined in step 4).

Device:2 has provided an optional AutonomousTransferCompletePolicy object, which allows an ACS to configure the CPE when to send the CWMP AutonomousTransferComplete PRC method. This version of the guidelines does not recommend the implementation of this feature. However, if a CPE Proxier chooses to implement this feature, the guidelines below can be followed:

- If the policy is disabled (Enable=0), then the CPE Proxier does not need to track the SMS events and does not need to trigger the CWMP AutonomousTransferComplete RPC.

- When the policy is enabled (Enable=1), then only report Download (TransferTypeFilter is set to “Download” or “Both”), since “Upload” is not supported in UPnP DM.
- When the policy is enabled (Enable=1), then only report Firmware (ResultTypeFilter includes “1 Firmware Upgrade Image”), since other types are not supported in UPnP DM.
- Observe the ResultTypeFilter settings, CPE Proxier can determine “success” or “failure” by calling the UPnP SMS::GetOperationInfo() action (indicated by the value of returned OperationState argument).

4.8 UPnP Security

UPnP DM (v2) has an optional security feature by adopting the UPnP deviceProtection:1 Service (DPS) [8]. DPS provides a general protection mechanism for UPnP devices from potential attacks by malicious Control Points (CPs), by utilizing TLS for UPnP action interaction and Wi-Fi Protected Setup (WPS) for introducing a CP to a UPnP device. In addition, DPS defines Roles (Public, Basic, and Admin etc.) to be associated to CPs to differentiate their level of privileges of interacting with the DPS interfaces. A CP with Admin Role is authorized to invoke any action of the DPS, while CPs with Public Role are restricted from a certain set of actions.

Each UPnP Device Control Protocol (DCP) is responsible to define its recommend security policies (i.e. list of Roles and Role-Action mappings) when adding DPS support, depending on its security requirements. The UPnP DM services have used the DPS defined Roles (i.e. Public, Basic, and Admin), and added 2 new Roles: dm:ThirdPartyAdmin and dm:UserAdmin. Each of the UPnP DM service (i.e. the BasicManagement Service, the ConfigurationManagement Service, and the SoftwareManagement Service) has provided recommended mappings of actions with these Roles, but implementations are free to change the Role-Action mappings or define their own Roles.

A common practice for applying Roles to actions is, for example, actions that may impact device status or user experience require Admin role, e.g. UPnP BMS::Reboot() action, and actions of information retrieving purpose are open to all CPs, e.g. CMS::GetSupportedDataModels(). Refer to UPnP action definition sections of the BasicManagement Service [5], the ConfigurationManagement Service [6], and the SoftwareManagement Service [7] for complete sets of security polices for all actions of each DM service.

4.8.1 UPnP Device Protection Service Issues with CPE Proxier and Guidelines

UPnP DPS is basically built on self-signed certificates and requires WPS introduction process (user interaction also involved) in order to add a new CP to the trusted pool (Access Control List, ACL) of a UPnP device. And in fact introduction does not necessarily (it normally doesn’t) give a CP Admin Role. A user account with Admin Role would be required to log in with a CP so that it temporally gains Admin privilege, or a CP may be configured by an Admin account with Admin Role using DPS actions. To perform any of the UPnP DM operations a CPE Proxier would have to obtain the Admin Role from the proxied UPnP device, according to security polices defined by each UPnP DM service, in which case user interactions will be needed for introduction and Admin Role

acquisition. A worse case is that the user may have to be asked to manually associate a CPE Proxier (either its CP certificate ID or username) with Admin Role using the device's UI or through an Admin CP device. The following sections discuss possible solutions for assigning the Admin Role to the CPE Proxier.

4.8.1.1 Guidelines for Carrier Provisioned UPnP DM Device

A UPnP DM device may be supplied by the carrier, in which case implementations of both the CPE Proxier and the UPnP DM device could support automated Admin Role assignment.

- CPE Proxiers can be provisioned with a CA signed certificate and use it when communicating with proxied UPnP DM devices. When discovering a UPnP DM device with DPS, the CPE Proxier shall establish TLS connection with it and invoke the DPS::GetAssignedRoles() action to check the Roles it possesses.
- UPnP DM device implementations shall pre-install root certificate of the CA that signs certificates to CPE Proxiers. For the first time when establishing a TLS session with a CPE Proxier and receiving any DPS or DM action call (e.g. DPS::GetAssignedRoles()), the UPnP DM device shall check the CPE Proxier's certificate against the root CA certificate. When the authenticity of the CPE Proxier is confirmed, the UPnP DM device shall automatically add the CPE Proxier's CPID to its ACL and assign it the Admin Role.

The following figure illustrates the process for assigning Admin Role to CPE Proxier.

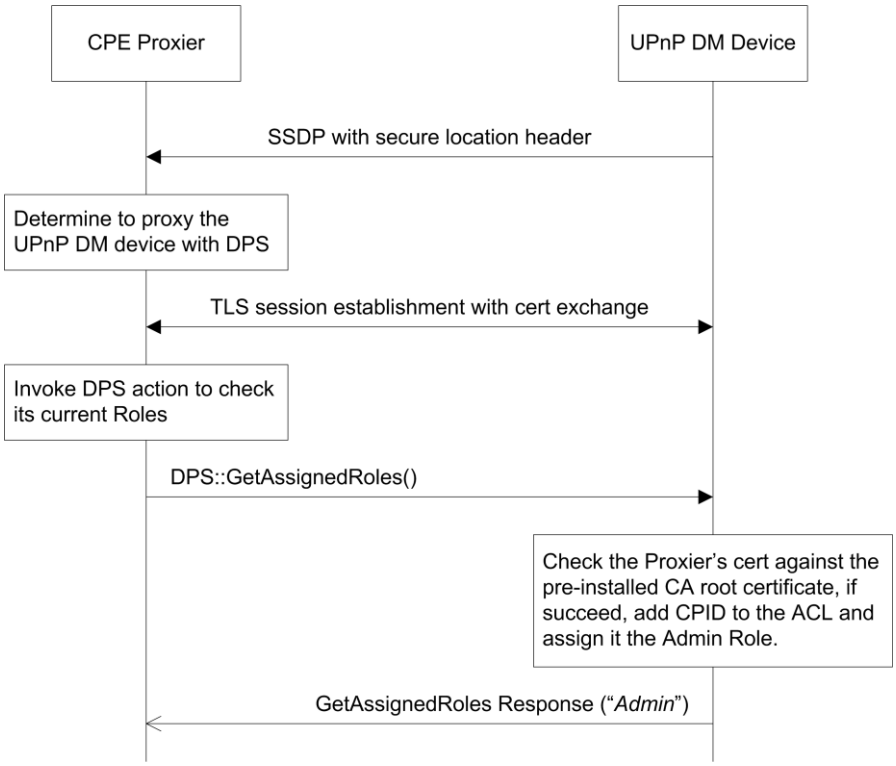

Figure 9 – Assigning the Admin Role to CPE Proxier

4.8.1.2 Guidelines for Retail Case

A retailed UPnP DM device is expected to comply with UPnP DM and DPS specifications, i.e. WPS introduction mechanism with self-signed certificate is implemented and UPnP DM security policies are applied, in which case user interaction would be inevitable. To reduce user interactions following guidelines could be considered by UPnP DM device and CPE Proxier implementations.

- Since UPnP DM has only mandated Admin and Public Roles for devices that implement DPS, it is recommended a CP being assigned the Admin Role after introduction process as a default policy.
- A CPE Proxier is often a function of device types like home gateways, which usually have no local GUI (except for web UI) nor directly connected to a display device. In this case supporting of Push Button Configuration (PBC) is suggested for both the CPE Proxier and UPnP DM device. Appendix A of DPS gives instructions on usage of PBC and section 3.3.2 gives an example flow.
    - When discovered and determined to proxy a UPnP DM device, the CPE Proxier could trigger the WPS introduction process and indicate use of PBC method according to DPS.
    - The UPnP DM device should timely prompt the user to push the button on the device by displaying a message or via other means.

End of Broadband Forum Technical Report TR-330

