TECHNICAL REPORT

TR-069
CPE WAN Management Protocol

Issue: 1 Amendment 6

Approval Date: March 2018

CWMP Version: 1.4

Issue History

Issue Number	Approval Date	Publication Date	Issue Editor	Changes
Issue 1	May 2004		Jeff Bernstein, 2Wire Tim Spets, Westell	Issue 1
Issue 1 Amendment 1	November 2006		Jeff Bernstein, 2Wire John Blackford, 2Wire Mike Digdon, SupportSoft Heather Kirksey, Motive William Lupton, 2Wire Anton Okmianski, Cisco	Clarification of original document
Issue 1 Amendment 2	November 2007		William Lupton, 2Wire Davide Moreo, Telecom Italia	CWMP Version 1.1: Multicast Download support, 10 AUTONOMOUS TRANSFER COMPLETE event, AutonomousTransferComplete method, additional Download fault codes, interoperability clarifications, minor editorial changes.
Issue 1 Amendment 3	November 2010		John Blackford, Pace Heather Kirksey, Alcatel-Lucent William Lupton, Pace	CWMP Version 1.2: Small updates for IPv6 related to DHCP, Additions for Software Module Management support (including new RPCs, Inform Event Codes, fault codes, and an Annex on UUIDs), ScheduleDownload RPC, and CancelTransfer RPC.
Issue 1 Amendment 4	July 2011		Sarah Banks, Cisco Andrea Colmegna, FASTWEB Tim Spets, Motorola Mobility	CWMP Version 1.3: Added Proxy management support and added Annex J and Appendix I. Table 4 Session timeout updated. Removed xsd Section A.6. Added Alias-Based Addressing additions, Section 3.6.1, Appendix II, and RPC Definition updates.
Issue 1 Amendment 5	11 November 2013	8 January 2014	John Blackford, Pace Mike Digdon, Aptean	CWMP Version 1.4: Added XMPP Connection Request support (Annex K and Appendix III), CPE standby-related behaviors (Annex L), UDP Lightweight Notification support (Annex M), and several other small clarifications/enhancements.
Issue 1 Amendment 6	16 March 2018	17 April 2018	KlausWich, Huawei John Blackford, Arris	CWMP Version 1.4: Added security enhancements, XMPP clarifications, Instance wildcards, Hearbeat event handling, Firmware Image support, XMPP over WebSockets for Connection Requests, HTTP Bulk Data Collection (from TR-157), Software Module Management (from TR-157) and several other clarifications / enhancements

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

Editors	Klaus Wich	Huawei
	John Blackford	ARRIS
Broadband User Services Work Area Directors	John Blackford	ARRIS
	Jason Walls	QACafe

Table of Contents

List of Tables

List of Figures

Executive Summary

A protocol for communication between a CPE and Auto-Configuration Server (ACS) that encompasses secure auto-configuration as well as other CPE management functions within a common framework.

1 Introduction

Note – Sections 1 and 2 of this document are introductory and do not define requirements of this protocol.

TR-069 describes the CPE WAN Management Protocol, intended for communication between a CPE and Auto-Configuration Server (ACS). The CPE WAN Management Protocol defines a mechanism that encompasses secure auto-configuration of a CPE, and also incorporates other CPE management functions into a common framework.

This document specifies the generic requirements of the management protocol methods which can be applied to any TR-069 CPE. Other documents specify the managed objects, or data models, for specific types of devices or services.

1.1 Functional Components

The CPE WAN Management Protocol is intended to support a variety of functionalities to manage a collection of CPE, including the following primary capabilities:

- Auto-configuration and dynamic service provisioning
- Software/firmware image management
- Software module management
- Status and performance monitoring
- Diagnostics

1.1.1 Auto-Configuration and Dynamic Service Provisioning

The CPE WAN Management Protocol allows an ACS to provision a CPE or collection of CPE based on a variety of criteria.

The provisioning mechanism allows CPE provisioning at the time of initial connection to the broadband access network, and the ability to re-provision or re-configure at any subsequent time. This includes support for asynchronous ACS-initiated re-provisioning of a CPE.

The identification mechanisms included in the protocol allow CPE provisioning based either on the requirements of each specific CPE, or on collective criteria such as the CPE vendor, model, software version, or other criteria.

The protocol also provides optional tools to manage the CPE-specific components of optional applications or services for which an additional level of security is required to control, such as those involving payments. The mechanism for control of such applications and services is the Software Module Management mechanism as defined in A.4.1.10 (ChangeDUState RPC), A.4.2.3 (DUStateChangeComplete RPC), and described in Appendix II / TR-157 Amendment 3 [32].

The provisioning mechanism allows straightforward future extension to allow provisioning of services and capabilities not yet included in this version of the specification.

1.1.2 Software/Firmware Image Management

The CPE WAN Management Protocol provides tools to manage downloading of CPE software/firmware image files. The protocol provides mechanisms for version identification, file download initiation (ACS initiated downloads and optional CPE initiated downloads), and notification of the ACS of the success or failure of a file download.

1.1.3 Software Module Management

The CPE WAN Management Protocol enables an ACS to manage modular software and execution environments on a CPE. Capabilities provided include the ability to install, update, and uninstall software modules as well as notification to the ACS of success or failure of each action. The protocol also provides support to start and stop applications on the CPE, enable and disable execution environments, and inventory the software modules available on the device.

1.1.4 Status and Performance Monitoring

The CPE WAN Management Protocol provides support for a CPE to make available information that the ACS may use to monitor the CPE's status and performance statistics. It also defines a set of mechanisms that allow the CPE to actively notify the ACS of changes to its state.

1.1.5 Diagnostics

The CPE WAN Management Protocol provides support for a CPE to make available information that the ACS may use to diagnose and resolve connectivity or service issues as well as the ability to execute defined diagnostic tests.

1.2 Positioning in the End-to-End Architecture

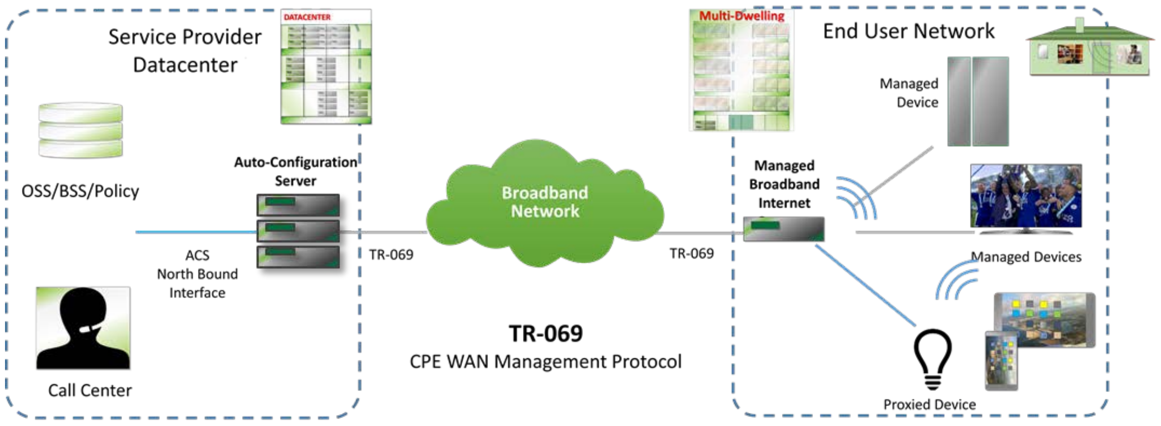
The ACS is a server that resides in the network and manages devices in or at the subscriber premises. The CPE WAN Management Protocol may be used to manage both DSL B-NTs and other types of CPE, including stand-alone routers and LAN-side client devices. It is agnostic to the specific access medium utilized by the service provider, although it does depend on IP-layer connectivity having been established by the device.

Note – in the case of a B-NT, TR-046 [2] describes the overall framework for B-NT auto-configuration, and TR-062 [3] and TR-044 [4] define the ATM layer and IP layer auto-configuration procedures. Other types of broadband CPE should make use of the protocols appropriate to their network architectures in order to obtain IP connectivity.

Note – where the CPE WAN Management Protocol is used to manage both a B-NT (or other Internet Gateway Device), and a LAN-side client device operating behind that B-NT (or other Internet Gateway Device), Annex F defines a mechanism to allow the ACS to associate the two so that they may be managed together.

Note – CPE Implementations might exist where the CPE WAN Management Protocol contains more than one CWMP Endpoint. Proxy management via the Virtual CWMP Device Mechanism is one such case.

Figure 1 – Positioning in the End-to-End Architecture



1.3 Security Goals

The CPE WAN Management Protocol is designed to provide a high degree of security. The security model is also designed to be scalable. It is intended to allow basic security to accommodate less robust CPE implementations, while allowing greater security for those that can support more advanced security mechanisms. In general terms, the security goals of the CPE WAN Management Protocol are as follows:

- Prevent tampering with the management functions of a CPE or ACS, or the transactions that take place between a CPE and ACS.
- Provide confidentiality for the transactions that take place between a CPE and ACS.
- Allow appropriate authentication for each type of transaction.
- Prevent theft of service.

1.4 Architectural Goals

The protocol is intended to provide flexibility in the connectivity model. The protocol is intended to provide the following:

- Allow both CPE and ACS initiated connection establishment, avoiding the need for a persistent connection to be maintained between each CPE and an ACS.
- The functional interactions between the ACS and CPE should be independent of which end initiated the establishment of the connection. In particular, even where ACS initiated connectivity is not supported, all ACS initiated transactions should be able to take place over a connection initiated by the CPE.
- Allow one or more ACSs to serve a population of CPE, which may be associated with one or more service providers.

The protocol is intended to support discovery and association of ACS and CPE:

- Provide mechanisms for a CPE to discover the appropriate ACS for a given service provider.
- Provide mechanisms to allow an ACS to securely identify a CPE and associate it with a user/customer. Processes to support such association should support models that incorporate user interaction as well as those that are fully automatic.

The protocol is intended to allow an ACS access to control and monitor various Parameters associated with a CPE. The mechanisms provided to access these Parameters are designed with the following premises:

- Different CPE may have differing capability levels, implementing different subsets of optional functionality. Additionally, an ACS may manage a range of different device types delivering a range of different services. As a result, an ACS must be able to discover the capabilities of a particular CPE.
- An ACS must be able to control and monitor the current configuration of a CPE.
- Other control entities besides an ACS may be able to control some Parameters of a CPE's configuration (e.g., via LAN-side auto-configuration). As a result, the protocol must allow an ACS to account for external changes to a CPE's configuration. The ACS should also be able to control which configuration Parameters can be controlled via means other than by the ACS.
- The protocol should allow vendor-specific Parameters to be defined and accessed.

The protocol is intended to minimize implementation complexity, while providing flexibility in trading off complexity vs. functionality. The protocol incorporates a number of optional components that come into play only if specific functionality is required. The protocol also incorporates existing standards where appropriate, allowing leverage of off-the-shelf implementations.

The protocol is intended to be agnostic to the underlying access network.

The protocol is also designed to be extensible. It includes mechanisms to support future extensions to the standard, as well as explicit mechanisms for vendor-specific extensions.

1.5 Assumptions

Some assumptions made in defining the CPE WAN Management Protocol are listed below:

- All CPE regardless of type (bridge[1], router, or other) obtain an IP address in order to communicate with an ACS.
- A CWMP Endpoint can interact with a single ACS at a time. At any time, a CWMP Endpoint is aware of exactly one ACS with which it can connect. (Note: a collection of ACSs behind a load balancer is considered a single ACS for the purposes of this document.)

[1] In the case of a bridge, the CPE must establish IP-layer connectivity specifically for management communication. The mechanism used to establish this connectivity would depend on the specific network architecture. For example, a DSL bridge may connect using IPoE with DHCP for address allocation, or may connect using PPPoE.

1.6 Terminology

The following terminology is used throughout the series of documents defining the CPE WAN Management Protocol.

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; e.g. Install, Update, Uninstall, Start, Stop, etc. (see Appendix II/TR-157 [32])
Applied	A change to the *CPE's* configuration has been Applied when the *CPE* has stopped using the previous configuration and begun using the new configuration.
B-NT	Broadband-Network Termination. A specific type of Broadband *CPE* used in DSL networks.
Committed	A change to the *CPE*'s configuration has been Committed when the change has been fully validated, the new configuration appears in the configuration *Data Model* for subsequent *ACS* operations to act on, and the change will definitely be *Applied* in the future, as required by the protocol specification.
Connection Request	The *ACS* action of requesting a *CPE* to start a *CWMP Session*, as defined in "3.2.2 ACS Connection Initiation."
CPE	Customer Premises Equipment; refers to a TR-069-compliant device and therefore covers both *Internet Gateway Devices* and LAN-side end devices. A CPE contains at least one *CWMP Endpoint*.
CPE Proxier	A *CPE* that is capable of proxying the communication between an *ACS* and a *Proxied Device*. There are two strategies for proxy management: *Virtual CWMP Device Mechanism* and *Embedded Object Mechanism*.
CR-Aware Standby	A *CPE* in CR-Aware Standby is able to receive *Connection Requests* and to accept or reject them.
CWMP	*CPE* WAN Management Protocol (the subject of this standard).
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 *Parameter*s that define the managed *Object*s accessible via TR-069 for a particular *Device* or service.
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.
Device	Used interchangeably with *CPE*.

DT Instance	Device Type Schema instance document. This is an XML document that conforms to the *DT Schema* and to any additional rules specified in or referenced by the *DT Schema*.
DT Schema	Device Type Schema. This is the XML Schema that is used for describing a Device's supported data model (see Annex B/TR-106 [16]).
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*.
Event	An indication that something of interest has happened that requires the *CPE* to notify the *ACS*.
Execution Environment	A software platform that enables the dynamic loading and unloading of *Software Modules*. Typical examples include Linux, OSGi, .NET, and Java ME. Some Execution Environments enable the sharing of resources amongst modules.
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 Unit*s. The following list of concepts could be considered an Execution Unit: services, scripts, software components, libraries, etc.
Forced Inform Parameter	A *Parameter* whose definition requires it to be included with every Inform *RPC*.
HEARTBEAT Event	A type of Event, reported using the Inform message, that occurs at regular intervals.
Instance Alias	A writeable string that uniquely identifies an instance within a *Multi-Instance Object*.
Instance Identifier	A value that uniquely identifies an instance within a *Multi-Instance Object*. It is either an *Instance Number* or an *Instance Alias*.
Instance Number	A read-only positive integer (>=1) that uniquely identifies an instance within a *Multi-Instance Object*.
Internet Gateway Device	A *CPE* device, typically a broadband router, that acts as a gateway between the WAN and the LAN.
Multi-Instance Object	An *Object* that can have multiple instances, all of which have the same structure and are located at the same level within the name hierarchy. Each instance is identified by an *Instance Identifier*.
Non-HEARTBEAT Inform	A regular Inform message triggered by any kind of event(s) except an HEARTBEAT Event.

Object	An internal node in the name hierarchy, i.e., a node that can have Object or *Parameter* children. An *Object* name is a *Path Name*.
Parameter	A name-value pair that represents part of a *CPE*'s configuration or status. A Parameter name is a *Path Name*.
Path Name	A name that has a hierarchical structure similar to files in a directory, with each level separated by a "." (dot). References an *Object* or a *Parameter*.
Partial Path Name	A *Path Name* that ends with a "." (dot). References an *Object* and represents a subset of the name hierarchy.
Periodic Contact	A time when the *CPE* has to send a *Periodic Inform*.
Periodic Inform	An Inform message including the "2 PERIODIC" event.
Proxied Device	An endpoint that communicates indirectly with an *ACS* via a *CPE Proxier*.
RPC	Remote Procedure Call.
Scheduled Contact	A specific time, resulting from a previous *ACS* request, when the *CPE* has to send an Inform. This does not include *Periodic Informs*.
Seen Missing	A *CPE* has been Seen Missing by the *ACS* when that *CPE* has missed a *Periodic Contact* or *Scheduled Contact* or when the *ACS* has failed to get a response to a *Connection Request*. A *CPE* cannot determine whether it has been Seen Missing.
Session	A contiguous sequence of CWMP *Transactions* between a *CWMP Endpoint* and an *ACS*. Note that a *Session* may span multiple TCP connections.
Software Module	The common term for all software (other than firmware) that will be installed on an *Execution Environment*, including the concepts of *Deployment Unit*s and *Execution Units*.
STB	Set Top Box. This *CPE* contains Audio and Video decoders and is intended to be connected to Analog TV and / or Home Theaters.
Timer-Aware Standby	A *CPE* in Timer-Aware Standby is able to wake up to handle *Periodic Contacts* and *Scheduled Contacts*. Partial Timer-Aware Standby is possible (*CPE* waking up only for *Scheduled Contacts*, for example).
Transaction	A message exchange between a *CWMP Endpoint* and an *ACS* consisting of a single request followed by a single response, initiated either by the *CPE* or *ACS*.

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.*
VoIP Endpoint	A Voice over IP *CPE* that acts as the initiation/termination point for VoIP calls. Examples of Endpoints include VoIP phones and analog terminal adapters (ATAs).

1.7 Abbreviations

This Technical Report defines the following abbreviations:

ACL	Access control list
ACS	Auto-Configuration Server
ADSL	Asymmetric Digital Subscriber Line
AES	Advanced Encryption Standard
ASCII	American Standard Code for Information Interchange
ATA	Analog terminal adapter
ATM	Asynchronous Transfer Mode
BOOTP	Boot Strap Protocol
CGI	Common Gateway Interface
CN	Common Name
CPE	Customer Premise Equipment
CR	Connection Request
CSRF	Cross-site request forgery
CWMP	CPE WAN Management Protocol
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DSL	Digital Subscriber Line
DSM-CC	Digital storage media command and control
DT	Device Type
DU	Deployment Unit
EE	Execution Environment
EU	Execution Unit
FLUTE	File Delivery over Unidirectional Transport
FTP	File transfer Protocol
HMAC	Hash-based Message Authentication Code
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol over Secure Socket Layer
IANA	Internet Assigned Numbers Authority
ID	Identifier
IP	Internet Protocol
IPv6	Internet Protocol version 6
ISO	International Organization for Standardization
LAN	Local Area Network

LSB	Least significant bit
MD5	Message-Digest algorithm 5
NAT	Network Address Translation
NTP	Network Time Protocol
NT	Network Termination
OSGi	OSGi Alliance (former Open Services Gateway initiative)
OUI	Organizationally Unique Identifier
PKCS	Public Key Cryptography Standards
QoS	Quality of Service
RFC	Request for Proposal
RPC	Remote Procedure Call
RSA	Rivest, Shamir and Adleman (crypto system)
SASL	Simple Authentication and Security Layer
SFTP	SSH File Transfer Protocol
SHA1	Secure Hash Algorithm 1
SNMP	Simple Network Management Protocol
SNTP	Simple Network Time Protocol
SOAP	Simple Object Access Protocol
SSH	Secure Shell
SSL	Secure Socket Layer
STB	Set Top Box
STUN	Session Traversal Utilities for NAT
TCP	Transmission Control Protocol
TFTP	Tiny File transfer Protocol
TLS	Transport Layer Security
TLV	Type length value
TR	Technical Report
TTL	Time to Live
TV	Television
UDP	User Datagram Protocol
UPnP	Universal Plug and Play
UPnP DM	Universal Plug and Play Device Management
UPnP IGD	Universal Plug and Play Internet Gateway Device
URI	Uniform Resource Identifier
URL	Universal Resource Locator
URN	Uniform Resource Name
UTC	Coordinated Universal Time
UTF	Universal Multiple-Octet Coded Character Set Transformation Format
UUID	Universally Unique Identifier
VoIP	Voice over Internet Protocol
WAN	Wide Area Network
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
XSD	XML Schema
XSS	Cross-Site Scripting

1.8 Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [1].

The key word “DEPRECATED” refers to a protocol feature, e.g. an RPC Method or Event Type, that is defined and valid in the current version of the standard but is not strictly necessary, e.g. because another more powerful feature has been defined. Such features SHOULD NOT be used; they might be removed from the next major version of the protocol.

The key word “OBSOLETED” refers to a protocol feature, e.g. an RPC Method or Event Type, that meets the requirements for being DEPRECATED, and in addition is obsolete. Such protocol features MUST NOT be used; they might be removed from a later minor version of the protocol, without this being regarded as breaking backwards compatability rules.

2 Architecture

2.1 Protocol Components

The CPE WAN Management Protocol comprises several components that are unique to this protocol, and makes use of several standard protocols. The protocol stack defined by the CPE WAN Management Protocol is shown in Figure 2. A brief description of each layer is provided in Table 1. Note that the CPE and ACS must adhere to the requirements of the underlying standard protocols unless otherwise specified.

Figure 2 – Protocol stack

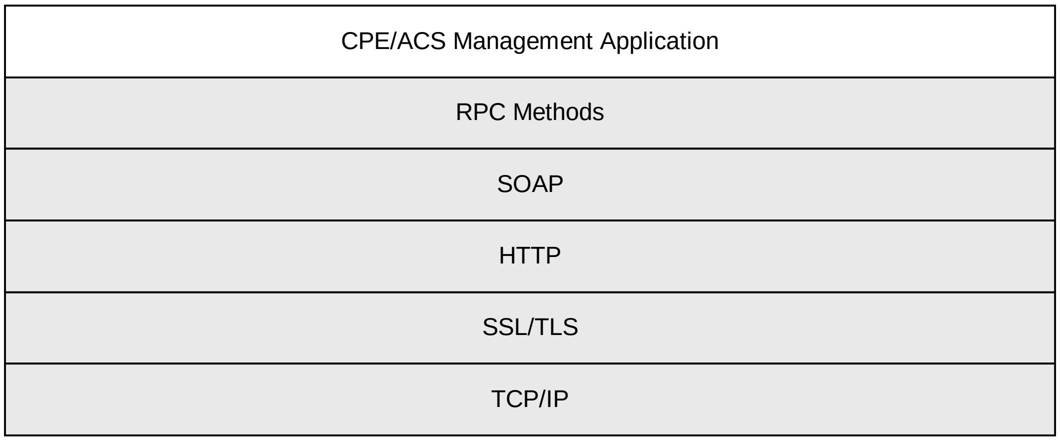


Table 1 – Protocol layer summary

Layer	Description
CPE/ACS Application	The application uses the CPE WAN Management Protocol on the CPE and ACS, respectively. The application is locally defined and not specified as part of the CPE WAN Management Protocol.
RPC Methods	The specific RPC methods that are defined by the CPE WAN Management Protocol. These methods are specified in Annex A.
SOAP	A standard XML-based syntax used here to encode remote procedure calls. Specifically SOAP 1.1, as specified in [12].
HTTP	HTTP 1.1, as specified in [6],[7].
TLS	The standard Internet transport layer security protocol. Specifically, TLS 1.2 (Transport Layer Security) as defined in [14] (or a later version). Note that previous versions of this specification referenced SSL 3.0 and TLS 1.0.
TCP/IP	Standard TCP/IP.

2.2 Security Mechanisms

The CPE WAN Management Protocol is designed to allow a high degree of security in the interactions that use it. The CPE WAN Management Protocol is designed to prevent tampering with the transactions that take place between a CPE and ACS, provide confidentiality for these transactions, and allow various levels of authentication.

The following security mechanisms are incorporated in this protocol:

- The protocol supports the use of TLS for communications transport between CPE and ACS. This provides transaction confidentiality, data integrity, and allows certificate-based authentication between the CPE and ACS.
- The HTTP layer provides an alternative means of CPE and ACS authentication based on shared secrets. Note that the protocol does not specify how the shared secrets are learned by the CPE and ACS.

2.3 Architectural Components

2.3.1 Parameters

The RPC Method Specification (see Annex A) defines a generic mechanism by which an ACS can read or write Parameters to configure a CPE and monitor CPE status and statistics. Parameters for various classes of CPE are defined in separate documents. At the time of writing the following standards define TR-069 data models.

- TR-098: Internet Gateway Device Data Model for TR-069 [27]
- TR-104: Provisioning Parameters for VoIP CPE [28]
- TR-135: Data Model for a TR-069 Enabled STB [29]
- TR-140: TR-069 Data Model for Storage Service Enabled Devices [30]
- TR-143: Enabling Network Throughput Performance Tests and Statistical Monitoring [31]
- TR-157: Component Objects for CWMP [32]
- TR-181: Device Data Model for TR-069 [34] and [35]
- TR-196: Femto Access Point Service Data Model [33]

Each Parameter consists of a name-value pair. The name identifies the particular Parameter. The value of a Parameter may be one of several defined data types (see TR-106 [16]).

Parameters may be defined as read-only or read-write. Read-only Parameters may be used to allow an ACS to determine specific CPE characteristics, observe the current state of the CPE, or collect statistics. Writeable Parameters allow an ACS to customize various aspects of the CPE's operation. All writeable Parameters must also be readable although those that contain confidential user information, e.g. passwords, may return empty values when read (this is specified in the corresponding data model definition). The value of some writeable Parameters may be independently modifiable through means other than the interface defined in this specification (e.g., some Parameters may also be modified via a LAN side auto-configuration protocol).

Because other protocols (as well as subscriber action) may independently modify the device configuration, the ACS cannot assume that it is the only entity modifying device configuration. Additionally, it is possible that a LAN-side mechanism could alter device configuration in such a way that it contravenes the intended ACS-supplied configuration. Care should be taken in the implementation of both WAN and LAN-side auto-configuration mechanisms, as well as subscriber-facing interfaces, to limit the instances of such an occurrence.

The protocol supports a discovery mechanism that allows an ACS to determine what Parameters a particular CPE supports, allowing the definition of optional Parameters as well as supporting straightforward addition of future standard Parameters.

The protocol also includes an extensibility mechanism that allows use of vendor-specific Parameters in addition to those defined in this specification.

2.3.2 File Transfers

The RPC Method Specification (see Annex A) defines mechanisms to facilitate file transfers for a variety of purposes, such as downloading firmware upgrades or vendor-specific configuration files, (optionally) installing or updating software modules, and (optionally) uploading configuration or log files from the device.

File transfers can be performed by means of Unicast or (for downloads) Multicast transport protocols. Unicast protocols include HTTP/HTTPS, FTP, SFTP and TFTP. Multicast protocols include FLUTE and DSM-CC. Support for HTTP/HTTPS is mandatory, and protocols other than those listed here can be supported.

When a file transfer is initiated by the ACS via any of the method calls that can cause a file transfer, the CPE is provided with the location of the file (or possibly files in the case of a software module installation or update) to be transferred, or details of the Multicast group to join (for Multicast downloads). The CPE then performs the transfer(s), and notifies the ACS of success or failure.

Downloads may be optionally initiated by a CPE. In this case, the CPE first requests a download of a particular file type from the ACS. The ACS may then respond by initiating the download following the same steps as an ACS-initiated download.

File transfers may also be optionally initiated by an external event, e.g. a Multicast firmware availability announcement or user-initiated software module updates. In this case, the CPE performs the transfer autonomously, and notifies the ACS of the success or failure.

2.3.3 CPE Initiated Sessions

The RPC Method Specification (see Annex A) defines a mechanism that allows a CPE to inform a corresponding ACS of various conditions, and to ensure that CPE-to-ACS communication will occur with some minimum frequency.

This includes mechanisms to establish communication upon initial CPE installation in order to ‘bootstrap’ initial customized Parameters into the CPE. It also includes a mechanism to establish periodic communication with the ACS on an ongoing basis, or when events occur that must be reported to the ACS (such as when the broadband IP address of the CPE changes).

In each case, when communication is established the CPE identifies itself uniquely via manufacturer and serial number information (and optional product class identifier) so that the ACS knows which CPE it is communicating with and can respond in an appropriate way.

2.3.4 Asynchronous ACS Initiated Sessions

An important aspect of service auto-configuration is the ability for the ACS to inform the CPE of a configuration change asynchronously. This allows the auto-configuration mechanism to be used for services that require near-real-time reconfiguration of the CPE. For example, this may be used to provide an end-user with immediate access to a service or feature they have subscribed to, without waiting for the next periodic contact.

The CPE WAN Management Protocol incorporates a mechanism for the ACS to issue a Connection Request to the CPE at any time, instructing it to establish a communication Session with the ACS.

While the CPE WAN Management Protocol also allows polling by the CPE in lieu of ACS-initiated connections, the CPE WAN Management Protocol does not rely on polling or establishment of persistent connections from the CPE to provide asynchronous notification.

The basic mechanism defined in the CPE WAN Management Protocol to enable asynchronous ACS initiated communication assumes direct IP addressability of the CPE from the ACS. An alternative mechanism is defined in Annex K, which accommodates CPE that are not directly addressable by the ACS.

3 Procedures and Requirements

This Section, along with the Annexes referenced in this Section, defines the normative requirements of the CPE WAN Management Protocol.

This Section also references a number of standards and other specifications that form part of the CPE WAN Management Protocol. Unless otherwise specified, the CPE and ACS MUST adhere to the requirements of these referenced specifications.

3.1 ACS Discovery

The CPE WAN Management Protocol defines the following mechanisms that MAY be used by a CPE to discover the address of its associated ACS:

1. The CPE MAY be configured locally with the URL of the ACS for each CWMP Endpoint. For example, this MAY be done via a LAN-side CPE auto-configuration protocol. If necessary, the CPE would use DNS to resolve the IP address of the ACS from the host name component of the URL.
2. As part of the IP layer auto-configuration, a DHCP server on the access network MAY be configured to include the ACS URL as a DHCP option [17] / [25] / [38]. If necessary, the CPE would use DNS to resolve the IP address of the ACS from the host name component of the URL. In this case additional DHCP options MAY be used to set:
    - The ProvisioningCode, which MAY be used to indicate the primary service provider and other provisioning information to the ACS.
    - The CWMPRetryMinimumWaitInterval, which MAY be used to set the initial value of the CWMP session retry minimum wait interval, as specified in Section 3.2.1.1.
    - The CWMPRetryIntervalMultiplier, which MAY be used to set the initial value of the CWMP session retry interval multiplier, as specified in Section 3.2.1.1.

    A CPE identifies itself to the DHCP server as supporting this method by including the string “dslforum.org” (all lower case) anywhere in the DHCPv4 Vendor Class Identifier (option 60), in the DHCPv4 V-I Vendor Class Option (option 124) or in a DHCPv6 Vendor Class (option 16) vendor-class-data item.

    The CPE MAY include either DHCPv4 option 43 or DHCPv4 option 125 (not both) in a DHCPv4 parameter request list (option 55) in order to indicate support for and to request option 43 or option 125. If the CPE does not use option 55 in this way, the server can assume that it supports and requests option 43 (not option 125). Similarly, the CPE MAY include DHCPv6 option 17 in a DHCPv6 Option Request Option (option 6).

    The CPE MAY use the values received from the DHCP server in the Vendor Specific Information (DHCPv4 option 43 / DHCPv4 option 125 / DHCPv6 option 17) to set the corresponding Parameters as listed in Table 2. If both DHCPv4 options are received, the CPE MUST use the DHCP option that it included in the DHCPv4 parameter request list (option 55); unless it did not use option 55, in which case the

CPE MUST use the DHCPv4 option 43 values. If both DHCPv4 and DHCPv6 options are received, the CPE MUST use the DHCPv6 option over the DHCPv4 option. This DHCP option is encoded as a list of one or more Encapsulated Vendor-Specific Options in the format defined in [17] / [25] / [38]. This list MAY include other vendor-specific options in addition to those listed here.

DHCP messages that include IANA Enterprise Numbers, i.e. DHCPv4 options 124 / 125 and DHCPv6 options 16 / 17, MUST use the Broadband Forum IANA Enterprise Number, which is 3561 decimal (the "ADSL Forum" entry in the IANA Private Enterprise Numbers registry [21]).

If the CPE obtained an ACS URL through DHCP and it cannot reach the ACS, the CPE MUST use DHCP to re-discover the ACS URL. The CPE MUST consider the ACS unreachable if it cannot establish a TCP connection to it for 300 seconds at each of the IP addresses to which the ACS URL resolves. If the CPE does not receive a DHCP reply, it MUST attempt to retry according to [23] / [38].

When the CPE needs to contact the ACS, it MUST use the DHCP discovery mechanism in the following scenarios:

- If the CPE has an empty value for the ManagementServer.URL Parameter, or
- If the CPE is unable to contact the ACS and the CPE originally (the first successful time after the most recent factory reset) obtained its ACS URL through DHCP.

This behavior enables the CPE to go back to the use of DHCP for finding the ACS if an ACS URL had not been pre-configured in the CPE. For example, this can handle the situation of setting an incorrect ACS URL on the CPE. This behavior is not meant as an ACS failover mechanism.

The CPE MUST remember the mechanism it used to locate the ACS after each factory reset. If the CPE did not use DHCP to discover the ACS URL, then it SHOULD NOT fall back to using DHCP for ACS discovery. If the CPE originally used DHCP for ACS discovery, then when it fails to contact the ACS, it MUST perform re-discovery via DHCP. The last requirement holds even if the ACS URL has been subsequently set through a non-DHCP mechanism.

Table 2 – Encapsulated Vendor Specific Options

Encapsulated Option	Encapsulated Vendor-Specific Option number			Parameter[2]
	DHCPv4 Option 43	DHCPv4 Option 125	DHCPv6 Option 17	
URL of the ACS	1	11	1	ManagementServer.URL
Provisioning code	2	12	2	DeviceInfo.ProvisioningCode
CWMP retry minimum wait interval	3	13	3	ManagementServer.CWMPRetryMinimumWaitInterval
CWMP retry interval multiplier	4	14	4	ManagementServer.CWMPRetryIntervalMultiplier

2 As defined in [27], [34], and [35].

All the encapsulated option values MUST be represented as strings and MUST be valid values for their corresponding Parameters. The specified URL MUST be an absolute URL. The encapsulated option values MUST NOT be null terminated. If the CPE receives an encapsulated option value that is null terminated, the CPE MUST accept the value provided, and MUST NOT interpret the null character as part of the value.

3. The CPE MAY have a default ACS URL that it MAY use if no other URL is provided to it.

The ACS URL MUST be in the form of a valid HTTP or HTTPS URL [6]. Use of an HTTPS URL indicates that the CPE MUST establish an SSL or TLS connection to the ACS.

Once the CPE has established a connection to the ACS via a CWMP Endpoint, the ACS MAY at any time modify the ACS URL Parameter stored within the CPE (Management-Server.URL, as defined in [27], [34], and [35]). Once modified, the CPE MUST use the modified URL for all subsequent connections to the ACS.

The "host" portion of the ACS URL is used by the CPE for validating the certificate from the ACS when using certificate-based authentication. Because this relies on the accuracy of the ACS URL, the overall security of this protocol is dependent on the security of the ACS URL.

The CPE SHOULD restrict the ability to locally configure the ACS URL to mechanisms that require strict security. The CPE MAY further restrict the ability to locally set the ACS URL to initial setup only, preventing further local configuration once the initial connection to an ACS has successfully been established such that only its existing ACS is permitted subsequently to change this URL. The CPE SHOULD prevent attempts to change the ACS URL from Cross Site Scripting [54] and Cross Site Request Forgery [55] attacks.

The use of DHCP for configuration of the ACS URL SHOULD be limited to situations in which the security of the link between the DHCP server and the CPE can be assured by the service provider. Since DHCP does not itself incorporate a security mechanism, other means of ensuring this security SHOULD be provided.

The ACS URL MAY contain a DNS hostname or an IP address. When resolving the ACS hostname, the DNS server might return multiple IP addresses. In this case, the CPE SHOULD randomly choose an IP address from the list. When the CPE is unable to reach the ACS, it SHOULD randomly select a different IP address from the list and attempt to contact the ACS at the new IP address. This behavior ensures that CPEs will balance their requests between different ACSs if multiple IP addresses represent different ACSs.

The CPE MUST NOT cache the DNS server response beyond the duration of time to live (TTL) returned by DNS server unless it cannot contact the DNS server for an update. This behavior is required by DNS RFC 1034 [5] and provides an opportunity for the DNS server to update stale data.

It is further RECOMMENDED that the CPE implements affinity to a particular ACS IP address. Affinity to a given IP address means that the CPE will attempt to use the same

IP address for as long as it can contact the ACS at this address. This creates a more stable system and can allow the ACS to perform better due to better caching. To implement the affinity the CPE SHOULD store to persistent storage the last successfully used IP address and the list of IP addresses from which it was selected. The CPE SHOULD continue to perform DNS queries as normal, but SHOULD continue using the same IP address for as long as it can contact the ACS and for as long as the list of IP addresses returned by the DNS does not change. The CPE SHOULD select a new IP address whenever the list of IP addresses changes or when it cannot contact the ACS. This provides an opportunity for service providers to reconfigure their network.

Port 7547 has been assigned by IANA for the CPE WAN Management Protocol (see [20]), and the ACS MAY use this port in its URL.

3.2 Connection Establishment

3.2.1 CPE Connection Initiation

The CPE MAY at any time initiate a connection to the ACS via a CWMP Endpoint using the pre-determined ACS address (see Section 3.1). A CPE MUST establish a connection to the ACS and issue the Inform RPC method (following the procedures described in Section 3.7.1.1) under the following conditions:

- The first time the CPE establishes a connection to the access network on initial installation
- On power-up or reset
- Once every ManagementServer.PeriodicInformInterval (for example, every 24 hours)
- When so instructed by the optional ScheduleInform method
- Whenever the CPE receives a valid Connection Request from an ACS (see Section 3.2.2)
- Whenever the URL of the ACS changes
- Whenever a Parameter is modified that is required to initiate an Inform on change.
- Whenever the value of a Parameter that the ACS has marked for “Active notification” via the SetParameterAttributes method is modified by an external cause (a cause other than the ACS itself). Parameter changes made by the ACS itself via SetParameterValues MUST NOT cause a new Session to be initiated. If a Parameter is modified more than once before the CPE is able to initiate a Session to perform the notification, the CPE MUST perform only one notification.

  If a Parameter is modified by an external cause while a Session is in progress, the change causes a new Session to be established after the current Session is terminated (it MUST NOT affect the current Session).

  In order to avoid excessive traffic to the ACS, a CPE MAY place a locally specified limit on the frequency of Parameter change notifications. This limit SHOULD be defined so that it is exceeded only in unusual circumstances. If this limit is exceeded, the CPE MAY delay by a locally specified amount initiation of a Session to notify the ACS. After this delay, the CPE MUST initiate a Session to the ACS and indicate all relevant Parameter changes (those Parameters that have been marked for notification) that have occurred since the last such notification.

- Whenever a download or upload completes (either successfully or unsuccessfully), provided that CPE policy indicates that the ACS needs to be notified of the download or upload completion.

  The ACS MUST always be notified of the completion of downloads or uploads that were specifically requested by the ACS.

  CPE policy MUST determine whether to notify the ACS of the completion of downloads or uploads that were not specifically requested by the ACS.

  Note – this CPE policy is remotely configurable via the parameters defined within the ManagementServer.AutonomousTransferCompletePolicy object. For example, the CPE might be configured to notify the ACS only if a download or upload (not requested by the ACS) was not completed successfully.

- Whenever an unsuccessfully terminated Session is retried according to the session retry policy specified in Section 3.2.1.1.

The CPE MUST NOT maintain an open connection to the ACS when no more outstanding messages exist on the CPE or ACS. Refer to Section 3.7.1.4 for details of CPE Session termination criteria.

3.2.1.1 Session Retry Policy

A CPE MUST retry failed Sessions to attempt to redeliver events that it has previously failed to deliver and to allow the ACS to make additional requests in a timely fashion. Section 3.7.1.5 details the rules for successful event delivery, for retrying event delivery, and for discarding events after failing to deliver them. The CPE MUST keep track of the number of times it has attempted to retry a failed Session.

If the CPE fails to establish a Session, this might be because the CPE supports CPE WAN Management Protocol v1.1 (or later) and the ACS supports only v1.0. If this situation is suspected (see Section 3.7.2.1), the CPE MUST revert to v1.0 when retrying the failed Session.

A CPE MUST retry a failed Session after waiting for an interval of time specified in Table 3 or when a new event occurs, whichever comes first. The CPE MUST choose the wait interval by randomly selecting a number of seconds from a range given by the post-reboot session retry count. When retrying a failed Session after an intervening reboot, the CPE MUST reset the wait intervals it chooses from as though it were making its first session retry attempt. In other words, if a Session is retried when a new event other than BOOT occurs, it does not reset the wait interval, although the continued occurrence of new events might cause Sessions to be initiated more frequently than shown in the table. Regardless of the reason a previous Session failed or the condition prompting session retry, the CPE MUST communicate to the ACS the session retry count.

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[3]	Default[4]	Data Model Parameter Name
Minimum wait interval	m	5 seconds	ManagementServer.CWMPRetryMinimumWaitInterval
Interval multiplier	k	2000	ManagementServer.CWMPRetryIntervalMultiplier

The factory default values of these Parameters MUST be the values that were hard-coded in previous versions of the CPE WAN Management Protocol, i.e. the values from the Default column. These values MAY be overridden by values obtained via DHCP, as explained in Section 3.1. They MAY also be changed by the ACS at any time.

Beginning with the tenth post-reboot session retry attempt, the CPE MUST choose from the fixed maximum range shown in Table 3. The CPE MUST continue to retry a failed Session until it is successfully terminated or until the rules defined in the "Retry/Discard Policy" column within Table 8 take precedence. Once a Session terminates successfully, the CPE MUST reset the session retry count to zero and no longer apply session retry policy to determine when to initiate the next Session.

Table 3 – Session Retry Wait Intervals

Post Reboot Session 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.$(k/1000)^4$
#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.$(k/1000)^{10}$

3.2.1.2 Use of random source port

Each time the CPE first connects to the ACS after rebooting, it SHOULD use a different ephemeral TCP source port in order to avoid the possibility of reusing the same port that it used last time. Reuse of the same port could cause the ACS to reject the connection if the elapsed time since the previous connection is less than the ACS's configured TCP TIME_WAIT value.

In order to minimize the probability that the same ephemeral port number is used on successive occasions, the port SHOULD be selected using a strong randomization mechanism.

[3] These symbols are used in Table 3.
[4] These are the values that were hard-coded in previous versions of the CPE WAN Management Protocol.

3.2.2 **ACS Connection Initiation**

The ACS MAY at any time request that a CWMP Endpoint initiate a connection to the ACS using the Connection Request mechanism. Support for this mechanism is REQUIRED in a CPE, and is RECOMMENDED in an ACS.

This mechanism relies on the CPE being reachable by the ACS. If the CPE is behind a firewall or if there is a NAT device between the ACS and CPE, the ACS might not be able to access the CPE at all. Annex K defines a mechanism that allows an ACS to contact a CPE that is not directly reachable by the ACS.

This mechanism relies on the ACS having had at least one prior communication with the CWMP Endpoint in a CPE-initiated interaction. During this interaction, if the ACS wishes to allow future ACS-initiated transactions, it would use the value of the ManagementServer.ConnectionRequestURL Parameter (see [27], [34] , and [35]). If the URL used for management access changes, the CPE MUST notify the ACS via an Inform message.

Port 7547 has been assigned by IANA for the CPE WAN Management Protocol (see [20]), and the CPE MAY use this port in the Connection Request URL.

3.2.2.1 Generic Connection Request Requirements

The Connection Request mechanism has the following generic requirements (meaning that these requirements apply to any Connection Request communications independent of the transport protocol being utiilzed):

- The CPE MUST accept Connection Requests from any source that has the correct authentication parameters for the target CPE.
- The CPE SHOULD restrict the number of Connection Requests for a particular CWMP Endpoint that it accepts during a given period of time in order to further reduce the possibility of a denial of service attack. If the CPE chooses to reject a Connection Request for this reason, the CPE MUST respond to that Connection Request with a transport specific error.
- If the CPE successfully authenticates and responds to a Connection Request for a particular CWMP Endpoint as described above, and if it is not already in a Session for the requested CWMP Endpoint, then it MUST, within 30 seconds of sending the response, attempt to establish a Session with the pre-determined ACS address (see Section 3.1) in which it includes the “6 CONNECTION REQUEST” EventCode in the Inform.

  Note – in practice there might be exceptional circumstances that would cause a CPE to fail to meet this requirement on rare occasions.
- If the ACS receives a successful response to a Connection Request but after at least 30 seconds the CPE has not successfully established a Session that includes the “6 CONNECTION REQUEST” EventCode in the Inform, the ACS MAY retry the Connection Request to that CPE.
- If the CPE, once it successfully authenticates and responds to a Connection Request, but before it establishes a Session to the ACS, receives one or more successfully authenticated Connection Requests for the same CWMP Endpoint, the CPE MUST

return a successful response for each of those Connection Requests, but MUST NOT initiate any additional Sessions for the same CWMP Endpoint as a result of these additional Connection Requests, regardless of how many it receives during this time.

- If the CPE is already in a Session with the ACS with at least one CWMP Endpoint when it receives one or more Connection Requests, it MUST NOT terminate any Session against any CWMP Endpoint prematurely as a result. The CPE MUST instead take a transport specific alternative action.

  This requirement holds for Connection Requests received any time during the interval that the CPE considers itself in a Session with at least one CWMP Endpoint, including the period in which the CPE is in the process of establishing the Session.

- The CPE MUST NOT reject a properly authenticated Connection Request for any reason other than those described above. If the CPE rejects a Connection Request for any of the reasons described above, it MUST NOT initiate a Session with the ACS as a result of that Connection Request.

3.2.2.2 HTTP Specific Connection Request Requirements

The Connection Request mechanism also has the following requirements when traversing over an HTTP transport:

- The Connection Request port MUST only be used for TR-069 connection requests. It MUST NOT be shared with any other protocol.
- The Connection Request MUST use an HTTP 1.1 GET to a specific URL designated by the CPE. The URL value is available as read-only Parameter on the CPE. The path of this URL value SHOULD be randomly generated by the CPE so that it is unique per CPE.
- The Connection Request MUST make use of HTTP, not HTTPS. The associated URL MUST be an HTTP URL.
- No data is conveyed in the Connection Request HTTP GET. Any data that might be contained SHOULD be ignored by the CPE.
- The CPE MUST use HTTP authentication [8] with the HTTP Digest Authentication scheme defined in [9] to authenticate the ACS before proceeding—the CPE MUST NOT initiate a connection to the ACS due to an unsuccessfully authenticated request.
- The CPE’s response to a successfully authenticated Connection Request MUST use either a “200 (OK)” or a “204 (No Content)” HTTP status code. The CPE MUST send this response immediately upon successful authentication, prior to it initiating the resulting Session. The length of the message-body (Section 3.3/RFC 7230 [6]) in the HTTP response MUST be zero.
- The CPE SHOULD restrict the number of Connection Requests for a particular CWMP Endpoint that it accepts during a given period of time in order to further reduce the possibility of a denial of service attack. If the CPE chooses to reject a Connection Request for this reason, the CPE MUST respond to that Connection Request with an HTTP 503 status code (Service Unavailable). Furthermore, the CPE SHOULD NOT include the HTTP Retry-After header in the response.

- If the CPE is already in a Session with the ACS with at least one CWMP Endpoint when it receives one or more Connection Requests, it MUST NOT terminate any Session against any CWMP Endpoint prematurely as a result. The CPE MUST instead take one of the following HTTP-specific alternative actions:
  - Reject each Connection Request by responding with an HTTP 503 status code (Service Unavailable). In this case, the CPE SHOULD NOT include the HTTP Retry-After header in the response.
  - Following the completion of the CWMP Endpoint's current Session, initiate exactly one new Session at a time (regardless of how many Connection Requests had been received during the previous Session) in which it includes the "6 CONNECTION REQUEST" EventCode in the Inform. The Connection Requests that are not accepted MUST be rejected (with an HTTP 503 status code). If the new Session is for the CWMP Endpoint currently in Session, the CPE MUST initiate the Session immediately after the existing Session is complete and all changes from that Session have been applied.
  - If the Connection Request is not for any CWMP Endpoint currently in Session, the CPE MAY initiate a new Session with the requested CWMP Endpoint while the existing Session is still active.

  This requirement holds for Connection Requests received any time during the interval that the CPE considers itself in a Session with at least one CWMP Endpoint, including the period in which the CPE is in the process of establishing the Session.
- If the HTTP connection request mechanism is disabled by the ACS, by setting the value of the Device.ManagementServer.HTTPConnectionRequestEnable parameter to "False", the CPE MUST close the port used for the HTTP connection request.

3.3 Use of TLS and TCP

Note – previous versions of this specification referenced SSL 3.0 and TLS 1.0. These are no longer mentioned in the text below, and SHOULD NOT be used.

The use of TLS to transport the CPE WAN Management Protocol 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 SHOULD support TLS 1.2 [14] (or a later version).
- If TLS 1.2 is supported, the CPE MUST support the IETF RFCs that update TLS 1. 2 [14].

- If TLS 1.2 is supported, the CPE SHOULD communicate its capabilities to the ACS as specified in Appendix E of RFC 5246 [14], as updated by RFC 7568 [57], allowing the ACS to choose the protocol.
- If TLS 1.2 is supported, the CPE SHOULD use the RFC 6066 [50] Server Name TLS extension to send the host portion of the ACS URL as the server name during the TLS handshake.

If TLS 1.2 is supported and the ACS URL has been specified as an HTTPS URL, the CPE MUST establish secure connections to the ACS, and SHOULD use TLS 1.2 (or, if supported, a later version).

Note – if the ACS 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.

- If TLS 1.2 is supported, the CPE MUST support the required (MUST/MUST NOT) recommendations and guidelines defined in the following sections of RFC 7525 [56]:
  - Section 3.1.1 – SSL/TLS Protocol Versions
  - Section 3.1.3 – Fallback to Lower Versions
  - Section 3.2 – Strict TLS
  - Section 3.3 – TLS Compression
  - Section 3.4 – TLS Session Resumption
  - Section 3.5 – TLS Renegotiation
  - Section 4.1 – Cipher Suites: General Guidelines
- A CPE MUST be able to initiate outgoing connections to the ACS.
- An ACS MUST be able to accept CPE-initiated connections.
- If TLS 1.2 (or a later version) is used, the CPE MUST authenticate the ACS using the ACS-provided certificate. Authentication of the ACS requires that the CPE MUST validate the certificate against a root certificate.
  - If the host portion of the ACS URL is a DNS name, this MUST be done according to the principles of RFC 6125 [49], using the host portion of the ACS URL as the reference identifier.
  - If the host portion of the ACS 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 [49].

  Note – wildcard certificates are permitted as described in RFC 6125 [49].

  Note – if the certificate contains any subjectAltName extensions whose values include DNS names, the CPE MUST (in accordance with RFC 6125 [49] Section 6.3) use these

DNS names in preference to any CN (Common Name) components of the certificate's Subject field when validating the 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 an ACS at a URL different from its pre-configured ACS URL, the CPE MUST validate the ACS certificate using the redirected ACS URL rather than the pre-configured ACS URL.

A CPE capable of obtaining absolute time SHOULD wait until it has accurate absolute time before contacting the ACS. If a CPE for any reason is unable to obtain absolute time, it can contact the ACS without waiting for accurate absolute time. If a CPE chooses to contact the ACS before it has accurate absolute time (or if it does not support absolute time), it MUST ignore those components of the ACS 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 OPTIONAL for both the CPE and ACS. Such client-side certificates MUST be signed by an appropriate chain. When client-side certificates are used to authenticate the CPE to the ACS, the Common Name (CN) field in the CPE certificate MUST be one of the following two types:
  - Unique CPE client certificate. In this case, the value of the CN field MUST be globally unique for each CPE. Specifically, the CN field MUST adhere to the format recommended for the username/userid in Section 3.4.4.

    Examples:

    ```
    00D09E-0123456789
    012345-STB-0123456789
    012345-Set%2DTop%2DBox-0123456789
    ```
  - Generic CPE client certificate. In this case, the value of the CN field MAY be the same among a set of CPE, such as all CPE of a specific model from a given vendor. The content of the CN field is not specified in this case.

    If generic CPE client certificates are used, the ACS SHOULD additionally authenticate the CPE using HTTP basic or digest authentication to establish the identity of a specific CPE.

3.4 Use of HTTP

SOAP messages are carried between a CPE and an ACS using HTTP 1.1 [6], where the CPE acts as the HTTP client and the ACS acts as the HTTP server.

Note – the CPE WAN Management Protocol also uses HTTP for Connection Requests, where the ACS acts as the HTTP client and the CPE acts as the HTTP server. This usage of HTTP is described in Section 3.2.2

3.4.1 Encoding SOAP over HTTP

The encoding of SOAP over HTTP extends the HTTP binding for SOAP, as defined in Section 6 of [12], as follows:

- A SOAP request from an ACS to a CPE is sent over an HTTP response, while the CPE's SOAP response to an ACS request is sent over a subsequent HTTP POST.
- When there is a SOAP response in an HTTP Request, or when there is a SOAP Fault response in an HTTP Request, the SOAPAction header in the HTTP Request MUST have no value (with no quotes), indicating that this header provides no information as to the intent of the message. That is, it MUST appear as follows:

  ```
  SOAPAction:
  ```

- When an HTTP Request or Response contains a SOAP Envelope, the HTTP Content-Type header MUST have a type/subtype of "text/xml".
- An empty HTTP POST MUST NOT contain a SOAPAction header.
- An empty HTTP POST MUST NOT contain a Content-Type header.
- An HTTP response that contains any CPE WAN Management Protocol payload (a SOAP request to the CPE, a successful SOAP response to the CPE, or a SOAP fault response containing a Fault element defined in Section 3.5) MUST use the HTTP status code 200 (OK).

When transporting a string value within an XML document, any characters which are special to XML MUST be escaped as specified by the XML specification [7]. Additionally, any characters other than printable ASCII characters, i.e. any characters whose decimal ASCII representations are outside the (inclusive) ranges 9-10 and 32-126, SHOULD be escaped as specified by the XML specification.

Below is an example HTTP Response from an ACS containing a SOAP Request:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: xyz

<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:cwmp="urn:dslforum-org:cwmp-1-0">
    <soap:Body>
        <cwmp:Request>
            <argument>value</argument>
        </cwmp:Request>
    </soap:Body>
</soap:Envelope>
```

Note – in the above example, the XML namespace prefixes used are only examples. The actual namespace prefix values are arbitrary, and are used only to refer to a namespace declaration.

Note – in the above example, the CWMP namespace identifier "urn:dslforum-org:cwmp-1-0" is only an example and is not necessarily the namespace that is defined by this specification.

3.4.2 Sessions

For a sequence of transactions forming a single Session, a CPE SHOULD maintain a TCP connection that persists throughout the duration of the Session. Specifically, the CPE MUST NOT initiate the closure of the TCP connection during the Session. However, if the TCP connection is cleanly closed after an HTTP request/response round trip, and if the Session has not otherwise terminated (either successfully or unsuccessfully) at the time of the last HTTP response, the CPE MUST continue the Session by sending the next HTTP request in a new TCP connection.

Note – use of a single TCP connection per Session is highly desirable, because re-negotiation of connections uses additional resources (network sockets, network bandwidth, processing power), which increases the chances of system overload and dropped Sessions, which in turn increases the load.

After receiving an authentication challenge, the CPE MUST send the next HTTP request (including the "Authorization" HTTP header) in the same TCP connection unless the ACS specifically requested, via a "Connection: close" HTTP header, that the TCP connection be closed[5]. In the latter case, the CPE MUST honor the ACS request, close the TCP connection, and send the next HTTP request (including the "Authorization" HTTP header) in a new TCP connection.

If the CPE for any reason fails to establish a TCP connection, fails to send an HTTP message, or fails to receive an HTTP response, the CPE MUST consider the Session unsuccessfully terminated. The CPE MUST wait a minimum of 30 seconds before declaring a failure to establish a TCP connection, or failure to receive an HTTP response.

The ACS SHOULD make use of a session cookie to maintain session state as described in [11]. The ACS SHOULD use only cookies marked for *Discard*, and SHOULD NOT assume that a CPE will maintain a cookie beyond the duration of the Session.

To ensure that an ACS can make use of a session cookie, a CPE MUST support the use of cookies as defined in [11] including the return of the cookie value in each subsequent HTTP POST, with the exception that a CPE need not support storage of cookies beyond the duration of a Session. The CPE MUST support the use of multiple cookies by the ACS, and MUST make available at least 512 bytes for storage of cookies.

Note – old-style "Netscape" cookies might be used when dealing with an ACS that utilizes a previous version of this specification.

When a Session is completed successfully or terminated unsuccessfully, a CPE MUST close the associated TCP connection to the ACS.

A CPE MUST support the use of HTTP redirection by the ACS. The CPE and ACS requirements associated with the use of HTTP redirection are as follows:

- A CPE MUST support the 302 (Found) and 307 (Temporary Redirect) HTTP status codes.

---

[5] This extra requirement is necessary because some ACS implementations might utilize the underlying TCP connection as a mechanism to detect replay attacks (see the note in Section 3.4.5). Such implementations would require the response to an authentication challenge to use the same TCP connection as the challenge.

- A CPE MAY also support the 301 (Moved Permanently) HTTP status code for redirection.
- The CPE MUST allow redirection to occur at any point during a Session (including the InformResponse), and the ACS MAY issue a redirect at any point during a Session.
- If the CPE is redirected, it MUST attempt to continue the Session using the URL provided in the HTTP redirect response. Specifically, the CPE MUST re-send the HTTP POST that resulted in the redirect response to the ACS at the redirected URL, and the CPE MUST then attempt to proceed with the Session exactly as if no redirection had occurred.
- If the CPE is redirected, the redirected URL MUST apply only to the remainder of the current Session or until a subsequent redirect occurs later in the same Session. The redirected URL MUST NOT be saved by the CPE (i.e. as the value of Management-Server.URL, as defined in [27], [34] , and [35]) for use in any subsequent Session or any subsequent retries of the Session. This requirement MUST hold even if the 301 (Moved Permanently) HTTP status code is used for redirection.
- The CPE MUST allow up to 5 consecutive redirections. If the CPE is redirected more than 5 times consecutively, it MAY consider the Session unsuccessfully terminated.
- The URL provided in HTTP redirection MAY be an HTTP or HTTPS URL. The appropriate transport mechanism (TCP or TLS) MUST be used with the new target regardless of the transport used before redirection.
- If TLS is used for the redirected Session, requiring the CPE to authenticate the ACS, the authentication MUST be based on the redirected URL rather than the pre-configured ACS URL (see Section 3.3).
- In an HTTP response sent by the ACS containing a redirect status code, the length of the HTTP entity-body (Section 3.3/RFC7230 [6]) MUST be zero. If the CPE receives an HTTP re-direct response with a non-empty entity-body, it MUST ignore the content of the entity-body.
- When redirected, the CPE MUST include all cookies associated with the Session in subsequent HTTP requests to the redirected ACS. The CPE MUST consider a redirect from an ACS to be a "verifiable transaction" as defined in [11], and thus it MUST send cookies to the redirected ACS without performing domain validation of each cookie.

3.4.3 File Transfers

If the CPE is instructed to perform a file transfer via a Download, ScheduleDownload, Upload, or ChangeDUState (Install or Update operations) request from the ACS, and if the file location is specified as an HTTP URL with the same host name as the ACS, then the CPE MUST choose one of the following approaches in performing the transfer:

- The CPE MAY send the HTTP GET/PUT over the already established connection. Once the file has been transferred, the CPE MAY then proceed in sending additional messages to the ACS while continuing to maintain the connection (this option is not valid for ScheduleDownload or ChangeDUState (Install or Update operations)).

- The CPE MAY open a second connection over which to transfer the file, while maintaining the Session to the ACS over which it can continue to send messages.
- The CPE MAY terminate the Session to the ACS and then perform the transfer.

If the file location is not an HTTP URL or is not in the same domain as the ACS or requires use of a different port, then only the latter two options are available to it.

A CPE MUST support the use of TLS as specified in Section 3.3 for establishment of a separate TCP connection to transfer a file using HTTP. The CPE MUST use TLS when the file location is specified as an HTTPS URL.

The CPE MUST support both HTTP basic and digest authentication for file transfers. The specific authentication method is chosen by the file server by virtue of providing a basic or digest authentication challenge. If authentication is used by the file server, the ACS MUST specify credentials using the specific RPC method used to initiate the transfer (i.e., Download, ScheduleDownload, Upload, ChangeDUState (Install or Update operations)).

3.4.4 Authentication

If the CPE is not authenticated using TLS, the ACS MUST authenticate the CPE using HTTP authentication [8]. If TLS is being used for encryption, the ACS SHOULD use the basic authentication scheme defined in [10]. If TLS is not being used, then the ACS MUST use the digest authentication scheme defined in [9].

The CPE MUST support both HTTP basic and digest authentication schemes. The ACS chooses the authentication scheme by virtue of providing a basic or digest authentication challenge. If TLS is being used for encryption, the CPE SHOULD preemptively send basic authentication credentials as defined [10].

Note – use of an authentication challenge requires the initial message (usually an Inform RPC method request) to be sent; use of preemptive basic authentication with TLS is secure and avoids the need for an additional request.

If the CPE has received an authentication challenge from the ACS (either basic or digest), the CPE SHOULD send an Authorization header in all subsequent HTTP requests for the duration of the TCP connection. Whether or not the CPE does this, the ACS MAY issue subsequent authentication challenges at any stage of the Session within a single or multiple TCP connections.

If any form of HTTP authentication is used to authenticate the CPE, the CPE SHOULD use a username/userid that is globally unique among all CPE manufacturers. Specifically, the CPE username/userid SHOULD be in one of the following two formats:

```
<OUI> "-" <ProductClass> "-" <SerialNumber>
<OUI> "-" <SerialNumber>
```

If a username/userid of the above format is used, the <OUI>, <ProductClass>, and <SerialNumber> fields MUST match exactly the corresponding Parameters included in the DeviceIdStruct in the Inform message, as defined in Annex A, except that, in order to guarantee that the Parameter values can be extracted from the username/userid, any character in the <ProductClass> and <SerialNumber> that is not 0-9, A-Z, a-z, or an underscore (“_”) MUST be escaped using URI percent-encoding, as defined in RFC 3986 [15].

Percent-encoding MUST be performed by converting each character to UTF-8 and then percent-encoding each octet. For example, the character é (LATIN SMALL LETTER E WITH ACUTE) is represented in UTF-8 as the two octets 0xC3 0xA9 and so would be percent-encoded as “%C3%A9”.

Note – prior to the clarification that conversion to UTF-8 occurs before percent-encoding, the escaped username/userid was ambiguous. For example, an implementation might have treated the character é as the ISO Latin-1 octet 0xE9, which would have been percent-encoded as “%E9”.

If a username/userid of the above format is used, the second form MUST be used if and only if the value of the ProductClass Parameter is empty.

Examples:

```
012345-0123456789
012345-STB-0123456789
012345-Set%2DTop%2DBox-0123456789
```

The password used in either form of HTTP authentication SHOULD be a unique value for each CPE. That is, multiple CPE SHOULD NOT share the same password. This password is a shared secret, and thus MUST be known by both CPE and ACS. The method by which a shared secret becomes known to both entities on initial CPE installation is outside the scope of this specification. Both CPE and ACS SHOULD take appropriate steps to prevent unauthorized access to the password, or list of passwords in the case of an ACS.

3.4.5 Digest Authentication

This Section outlines requirements for use of digest authentication within the CPE WAN Management Protocol. These requirements apply to authentication of connections for RPC exchanges as well as for file transfers. Note that ACS and CPE play the role of HTTP client and server interchangeably for different types of connections. The ACS plays the role of the HTTP client when making connection requests. The CPE plays the role of the HTTP client when initiating connections to the ACS.

The CPE and the ACS MUST support the RFC 2617 “qop” option containing the value “auth”. According to RFC 2617, this means that the HTTP client MUST use a new style digest mechanism when this option is provided to it by the HTTP server.

When using digest authentication, for each new TCP connection opened, the ACS SHOULD use a new nonce value and the CPE SHOULD use a new cnonce value.

Note – if TLS is not used for a CPE WAN Management Protocol Session, the policy used by the ACS for reusing nonce values for HTTP authentication can significantly affect the security of the Session. In particular, if the ACS re-uses a nonce value when re-authenticating across multiple TCP connections, the ACS can be vulnerable to replay attacks. However, if TLS is used for a Session, then this risk is largely mitigated.

The CPE and the ACS MUST support the MD5 digest algorithm. The CPE MUST additionally support the MD5-sess digest algorithm.

3.4.6 Additional HTTP Requirements

The following additional HTTP-related requirements are specified:

- Whenever the ACS sends an empty HTTP response, it MUST use the “204 (No Content)” HTTP status code.
- The CPE MUST NOT make use of pipelining as defined in HTTP 1.1 [6].

3.4.7 HTTP Compression

This section outlines requirements for the use of HTTP compression using the HTTP protocol to exchange content encodings as defined in section 4 “Transfer Codings” of RFC 7230 [6] within the CPE WAN Management Protocol. These requirements apply to compression of RPC exchanges as well as for file transfers.

The following additional HTTP-related requirements are specified:

- The ACS SHOULD support the exchange of content encodings as defined in section 4 “Transfer Codings” of RFC 7230 [6].
- The CPE SHOULD support the exchange of content encodings as defined in section 4 “Transfer Codings” of RFC 7230 [6].

In order for the CPE and ACS to efficiently exchange compressed messages the CPE MUST send the compressed message with the Content-Encoding header defined by the ManagementServer.HTTPCompression Parameter, unless that Parameter is set to “Disabled”.

If the ACS doesn’t support the Content-Encoding header or value, the ACS MUST respond with the “415 – Unsupported Media Type” HTTP status code. Upon receipt of the “415 – Unsupported Media Type” HTTP status code, the CPE MUST remove the compression and retry the session as defined in section 3.2.1.1 Session Retry Policy.

The ACS can enable HTTP compression by setting the ManagementServer.HTTP-Compression Parameter to a value supported by the CPE and ACS. The ACS can disable HTTP compression by setting the ManagementServer.HTTPCompression Parameter to “Disabled”. The CPE lists the supported HTTP compression mechanisms in the ManagementServer.HTTPCompressionSupported Parameter.

3.5 Use of SOAP

The CPE WAN Management Protocol defines SOAP 1.1 [12] as the encoding syntax to transport the RPC method calls and responses defined in Annex A.

The following describes the mapping of RPC methods to SOAP encoding:

- The encoding MUST use the standard SOAP 1.1 envelope and serialization namespaces:
    - Envelope namespace identifier "http://schemas.xmlsoap.org/soap/envelope/"
    - Serialization namespace identifier "http://schemas.xmlsoap.org/soap/encoding/"

- The data types used in Annex A correspond directly to the data types defined in the SOAP 1.1 serialization namespace. (In general, the types used in Annex A are restricted subsets of the corresponding SOAP types.)
- Following the SOAP specification [12], elements specified as being of type “anySimpleType” MUST include a type attribute to indicate the actual type of the element.
- Elements of a type other than “anySimpleType” MAY include a type attribute if and only if the element is defined using a named data type in the RPC method XML schema in Annex A. If a type attribute is included, the value of the type attribute MUST exactly match the named data type specified in the schema.
- For an array argument, the argument name specified in the table in which the array is defined MUST be used as the name of the overall array element. The name of the member elements of an array MUST be the data type of the array as specified in the table in which the array is defined (excluding the brackets and any length limitation given in parentheses), and MUST NOT be namespace qualified. For example, an argument named ParameterList, which is an array of ParameterValueStruct structures, would be encoded as:

```
<ParameterList soap-enc:arrayType="cwmp:ParameterValueStruct[2]">
    <ParameterValueStruct>
        <name>Parameter1</name>
        <value xsi:type="someType">1234</value>
    </ParameterValueStruct>
    <ParameterValueStruct>
        <name>Parameter2</name>
        <value xsi:type="someType">5678</value>
    </ParameterValueStruct>
</ParameterList>
```

  As a second example, the MethodList array in the GetRPCMethodsResponse would be encoded as:

```
<MethodList soap-enc:arrayType="xsd:string[3]">
    <string>GetRPCMethods</string>
    <string>Inform</string>
    <string>TransferComplete</string>
</MethodList>
```

Note – in the above examples, the XML namespace prefixes used are only examples. The actual namespace prefix values are arbitrary, and are used only to refer to a namespace declaration.

Note – it is always necessary to specify an XML namespace prefix for the arrayType attribute. For arrays of CWMP-specific types this will always be the CWMP namespace prefix, and for arrays of other types it will always be the XML Schema namespace prefix or the SOAP encoding namespace prefix.

- Regarding the SOAP specification for encoding RPC methods (Section 7 of [12]), for each method defined in Annex A, each argument listed in the method call represents an [in] parameter, while each argument listed in the method response represents an [out] parameter. There are no [in/out] parameters used.

- The RPC methods defined use the standard SOAP naming convention whereby the response message corresponding to a given method is named by adding the "Response" suffix to the name of the method.
- A SOAP Envelope MUST contain exactly one Body element.
- A CPE MUST be able to accept a SOAP request with a total envelope size of at least 32 kilobytes (32768 bytes) without resulting in a "Resources Exceeded" response.
- A CPE MUST be able to generate a SOAP response of any required length without resulting in a "Resources Exceeded" response, i.e. there is no maximum CPE SOAP response length.
- An ACS MUST be able to accept a SOAP request with a total envelope size of at least 32 kilobytes (32768 bytes) without resulting in a "Resources Exceeded" response.
- An ACS MUST be able to generate a SOAP response of any required length without resulting in a "Resources Exceeded" response, i.e. there is no maximum ACS SOAP response length.
- A fault response MUST make use of the SOAP Fault element using the following conventions:
    - The SOAP `faultcode` element MUST indicate the source of the fault, either "Client" or "Server", as appropriate for the particular fault. In this usage, "Client" represents the originator of the SOAP request, and "Server" represents the SOAP responder. The value of the SOAP `faultcode` element MUST either be unqualified or else be qualified with the SOAP envelope namespace prefix. The recipient of the fault response need not make use of the value of this element, and MAY ignore the SOAP `faultcode` element entirely.
    - The SOAP `faultstring` sub-element MUST contain the string "CWMP fault".
    - The SOAP `detail` element MUST contain a Fault structure. The RPC method XML schema in Annex A formally defines this structure. This structure contains the following elements:
        - A `FaultCode` element that contains a single numeric fault code as defined in Annex A.
        - A `FaultString` element that contains a human readable description of the fault.
        - A `SetParameterValuesFault` element, to be used only in an error response to the SetParameterValues method, that contains a list of one or more structures indicating the specific fault associated with each parameter in error. This structure contains the following elements:
            - A `ParameterName` element that contains the full Path Name of the Parameter in error.

- A `FaultCode` element that contains a single numeric fault code as defined in Annex A that indicates the fault associated with the particular Parameter in error.
- A `FaultString` element that contains a human readable description of the fault for the particular Parameter in error.

Below is an example envelope containing a fault response:

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:cwmp="urn:dslforum-org:cwmp-1-0">
    <soap:Header>
        <cwmp:ID soap:mustUnderstand="1">1234</cwmp:ID>
    </soap:Header>
    <soap:Body>
        <soap:Fault>
            <faultcode>Client</faultcode>
            <faultstring>CWMP fault</faultstring>
            <detail>
                <cwmp:Fault>
                    <FaultCode>9000</FaultCode>
                    <FaultString>Upload method not supported</FaultString>
                </cwmp:Fault>
            </detail>
        </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

Below is an example envelope containing a fault response for a SetParameterValues method call:

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:cwmp="urn:dslforum-org:cwmp-1-0">
    <soap:Header>
        <cwmp:ID soap:mustUnderstand="1">1234</cwmp:ID>
    </soap:Header>
    <soap:Body>
        <soap:Fault>
            <faultcode>Client</faultcode>
            <faultstring>CWMP fault</faultstring>
            <detail>
                <cwmp:Fault>
                    <FaultCode>9003</FaultCode>
                    <FaultString>Invalid arguments</FaultString>
                    <SetParameterValuesFault>
                        <ParameterName>
                            Device.Time.NTPServer1
                        </ParameterName>
                        <FaultCode>9007</FaultCode>
                        <FaultString>Invalid IP Address</FaultString>
                    </SetParameterValuesFault>
                    <SetParameterValuesFault>
                        <ParameterName>
                            Device.Time.LocalTimeZoneName
                        </ParameterName>
                        <FaultCode>9007</FaultCode>
                        <FaultString>String too long</FaultString>
                    </SetParameterValuesFault>
                </cwmp:Fault>
            </detail>
        </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

Note – in the above examples, the XML namespace prefixes used are only examples. The actual namespace prefix values are arbitrary, and are used only to refer to a namespace declaration.

Note – in the above example, the CWMP namespace identifier "urn:dslforum-org:cwmp-1-0" is only an example and is not necessarily the namespace that is defined by this specification.

A fault response MUST only be sent in response to a SOAP request. A fault response MUST NOT be sent in response to a SOAP response or another fault response.

If a fault response does not follow all of the above requirements, the SOAP message MUST be deemed invalid by the recipient. The consequences of invalid SOAP on the CPE WAN Management Protocol Session are described in Section 3.7.

- When processing a received envelope, both ACS and CPE MAY ignore: (a) any unknown XML elements within the SOAP Body[6] and their sub elements or content, (b) any unknown XML attributes and their values, (c) any embedded XML comments, and (d) any XML processing instructions. Alternatively the ACS and CPE MAY explicitly validate the received XML and reject an envelope that includes unknown elements. Note that this precludes extending existing messages by including additional arguments without changing the name of the message.
- If an RPC method requires references to XML Schema namespaces (for example for the "type" attribute, or for references to XML Schema data types), these references MUST be to the 2001 versions of these namespace definitions, specifically, http://www.w3.org/2001/XMLSchema-instance and http://www.w3.org/2001/XML-Schema. The recipient MAY reject an RPC method that references a different version of either of these namespaces.

As an example of an RPC method encoded as described above, a GetParameterNames request would be encoded as:

```
<soap-env:Envelope xmlns:soap-enc="http://schemas.xmlsoap.org/soap/encoding/"
                   xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/"
                   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xmlns:cwmp="urn:dslforum-org:cwmp-1-0">
    <soap-env:Header>
        <cwmp:ID soap-env:mustUnderstand="1">0</cwmp:ID>
    </soap-env:Header>
    <soap-env:Body>
        <cwmp:GetParameterNames>
            <ParameterPath>Object.</ParameterPath>
            <NextLevel>0</NextLevel>
        </cwmp:GetParameterNames>
    </soap-env:Body>
</soap-env:Envelope>
```

Note – in the above example, the XML namespace prefixes used are only examples. The actual namespace prefix values are arbitrary, and are used only to refer to a namespace declaration.

Note – the CWMP namespace prefix is specified only for elements that are defined at the top level of the CWMP schema (ID and GetParameterNames in the above example). It is incorrect to specify a namespace on elements contained within such elements (ParameterPath and NextLevel in the

[6] With the exception that reception of a SOAP request to invoke an unsupported RPC method MUST result in a SOAP-layer fault response with a fault code indicating "Method not Supported" (fault code 8000 or 9000).

above example). This is because the CWMP schema specifies an elementFormDefault value of "unqualified".

Note – in the above example, the CWMP namespace identifier "urn:dslforum-org:cwmp-1-0" is only an example and is not necessarily the version that is defined by this specification.

The CPE WAN Management Protocol defines a series of SOAP Header elements as specified in Table 4.

Table 4 – SOAP Header Elements

Tag Name	Description
ID	This header element MAY be used to associate SOAP requests and responses using a unique identifier for each request, for which the corresponding response contains the matching identifier. The value of the identifier is an arbitrary string and is set at the discretion of the requester. If used in a SOAP request, the ID header MUST appear in the matching response (whether the response is a success or failure). Because support for this header is required, the mustUnderstand attribute MUST be set to "1" (true) for this header.
HoldRequests	This header MAY be included in SOAP envelopes sent from an ACS to a CPE to regulate transmission of requests from the CPE. This header MUST NOT appear in envelopes sent from a CPE to an ACS. This tag has Boolean values of "0" (false) or "1" (true). If the tag is not present, this is interpreted as equivalent to a "0" (false). The behavior of the CPE on reception of this header is defined in Section 3.7.1.3. Support in the CPE for this header is REQUIRED. Because support for this header is required, the mustUnderstand attribute MUST be set to "1" (true) for this header. This header is DEPRECATED because it unnecessarily complicates the protocol and CWMP Session flow.
SessionTimeout	This header MAY be included in SOAP envelopes sent from a CPE to an ACS during CWMP Session initiation for the sole use of providing a suggestion of an acceptable CWMP Session timeout duration. This header MUST NOT appear in envelopes sent from an ACS to a CPE. This header also MUST NOT appear in envelopes whose SOAP body does not include a CWMP Inform request. The SessionTimeout is an integer that represents the number of seconds that SHOULD be used by the ACS as the amount of time to wait before timing out a CWMP Session due to the CPE not responding. The suggested SessionTimeout MUST be 30 seconds or greater. Because support for this header is OPTIONAL, the mustUnderstand attribute MUST be set to "0" (false) for this header.
SupportedCWMPVersions	This header MUST be included in SOAP envelopes sent from a CPE to an ACS during CWMP Session initiation for the sole purpose of providing to the ACS a list of supported CWMP versions. CPEs that support CWMP version 1.3 (or earlier) MUST NOT send this header. This header MUST NOT appear in envelopes sent from an ACS to a CPE. This header also MUST NOT appear in envelopes whose SOAP body does not include a CWMP Inform request. The SupportedCWMPVersions value is a comma-separated list of CWMP versions. Existing versions at the time of publication are 1.0, 1.1, 1.2, 1.3, and 1.4. Because support for this header is OPTIONAL, the mustUnderstand attribute MUST be set to "0" (false) for this header.
UseCWMPVersion	This header MUST be included in SOAP envelopes sent from an ACS to a CPE during CWMP Session initiation for the sole purpose of providing the chosen CWMP version to the CPE, if and only if the SupportedCWMPVersions header was included on the Inform request and the ACS supports the SupportedCWMPVersions header. This header MUST NOT appear in envelopes sent from a CPE to an ACS. This header also MUST NOT appear in envelopes whose SOAP body does not include a CWMP InformResponse. The UseCWMPVersion value MUST be a single version, chosen from the list in SupportedCWMPVersions that was sent with the Inform request. Because this header is only sent when the CPE sends SupportedCWMPVersions, the mustUnderstand attribute MUST be set to "1" (true) for this header.

Below is an example of two messages showing the use of some of the non-DEPRECATED headers:

CPE to ACS SOAP header

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:cwmp="urn:dslforum-org:cwmp-1-2">
    <soap:Header>
        <cwmp:ID soap:mustUnderstand="1">1234</cwmp:ID>
        <cwmp:SessionTimeout soap:mustUnderstand="0">40</cwmp:SessionTimeout>
    </soap:Header>
    <soap:Body>
        <cwmp:Action>
            <argument>value</argument>
        </cwmp:Action>
    </soap:Body>
</soap:Envelope>
```

ACS to CPE SOAP header

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:cwmp="urn:dslforum-org:cwmp-1-2">
    <soap:Header>
        <cwmp:ID soap:mustUnderstand="1">1234</cwmp:ID>
    </soap:Header>
    <soap:Body>
        <cwmp:Action>
            <argument>value</argument>
        </cwmp:Action>
    </soap:Body>
</soap:Envelope>
```

Note – in the above example, the XML namespace prefixes used are only examples. The actual namespace prefix values are arbitrary, and are used only to refer to a namespace declaration.

Note – in the above example, the CWMP namespace identifier "urn:dslforum-org:cwmp-1-2" is only an example and is not necessarily the version that is defined by this specification.

3.6 RPC Support Requirements

Table 5 provides a summary of all methods, and indicates the conditions under which implementation of each RPC method defined in Annex A is REQUIRED or OPTIONAL.

Table 5 – RPC message requirements

Method name	CPE requirement	ACS requirement
CPE methods	Responding	Calling
`GetRPCMethods`	REQUIRED	OPTIONAL
`SetParameterValues`	REQUIRED	REQUIRED
`GetParameterValues`	REQUIRED	REQUIRED
`GetParameterNames`	REQUIRED	REQUIRED
`SetParameterAttributes`	REQUIRED	OPTIONAL
`GetParameterAttributes`	REQUIRED	OPTIONAL
`AddObject`	REQUIRED	OPTIONAL
`DeleteObject`	REQUIRED	OPTIONAL
`Reboot`	REQUIRED	OPTIONAL
`Download`	REQUIRED[7]	REQUIRED[7]
`ScheduleDownload`	OPTIONAL	OPTIONAL
`Upload`	OPTIONAL	OPTIONAL
`FactoryReset`	OPTIONAL	OPTIONAL
`GetQueuedTransfers` **(DEPRECATED)**	OPTIONAL[8]	OPTIONAL
`GetAllQueuedTransfers`	OPTIONAL	OPTIONAL
`CancelTransfer`	OPTIONAL	OPTIONAL
`ScheduleInform`	OPTIONAL	OPTIONAL
`ChangeDUState`	OPTIONAL	OPTIONAL
`SetVouchers` **(DEPRECATED)**	OPTIONAL[9]	OPTIONAL
`GetOptions` **(DEPRECATED)**	OPTIONAL[9]	OPTIONAL
ACS methods	Calling	Responding
`GetRPCMethods`	OPTIONAL	REQUIRED
`Inform`	REQUIRED	REQUIRED
`TransferComplete`	REQUIRED[10]	REQUIRED[11]
`AutonomousTransferComplete`	OPTIONAL	REQUIRED
`DUStateChangeComplete`	OPTIONAL[12]	OPTIONAL[13]
`AutonomousDUStateChangeComplete`	OPTIONAL	OPTIONAL
`RequestDownload`	OPTIONAL	OPTIONAL
`Kicked` **(DEPRECATED)**	OPTIONAL	OPTIONAL[14]

7 REQUIRED only if file downloads of any type are supported.
8 DEPRECATED in favor of GetAllQueuedTransfers.
9 The voucher mechanism has been DEPRECATED in favor of the Software Module Management mechanism.
10 REQUIRED only if file downloads or uploads of any type are supported.
11 REQUIRED only if the ACS supports initiation of file downloads or uploads.
12 If the CPE responds to the ChangeDUState RPC then it MUST support this RPC.
13 If the ACS supports the ChangeDUState RPC then it MUST respond to this RPC.
14 DEPRECATED due to the deprecation of Annex D, which is the Section that defined the usage of this RPC.

3.6.1 Alias-Based Addressing Mechanism Requirements

The OPTIONAL Alias-Based Addressing Mechanism makes use of the Instance Alias identifiers defined in A.2.2.2 and described in Appendix II.

An ACS that supports the Alias-Based Addressing Mechanism MUST fully comply with all the following requirements:

1. An ACS MUST NOT use Instance Alias identifiers with a CPE that has not included the ManagementServer.AliasBasedAddressing Parameter (set to true) in the Inform Parameters.

A CPE that supports the Alias-Based Addressing Mechanism MUST fully comply with all the following requirements:

1. A CPE MUST support Instance Alias identifiers as alternative methods to address Multi-Instance Objects in addition to Instance Number identifiers.
2. Upon creating an instance of a Multi-Instance Object, the CPE MUST assign a unique Instance Alias (using a "cpe-" prefix) unless the Instance Alias value was provided in CWMP RPC from the ACS. Aliases for instances created as a result of any other action or contained in CPE factory defaults MUST be created with the “cpe-“ prefix. The CPE MUST use the same Instance Alias values for factory default objects across all instances of the CPE of the same hardware model and software version.
3. The CPE MUST support the ManagementServer.AliasBasedAddressing Parameter as a Forced Inform Parameter and include it (set to true) in all Inform messages.
4. The CPE MUST support write access to the ManagementServer.AutoCreate-Instances Parameter that is used by the ACS to enable or disable the CPE Auto-Create Instance Mechanism (defined in A.3.2.1).
5. The CPE MUST support write access to the ManagementServer.InstanceMode Parameter. This is used by the ACS to control whether the CPE will use Instance Numbers or Instance Aliases in returned Path Names as detailed in requirements 6, 7, 8 and 9.
6. Upon receiving a request, the CPE MUST support uniform or mixed Instance Identifiers for Objects in the Parameter Path Name. A mixed Parameter Path Name has different Instance Identifier types (Instance Number or Instance Alias) at the different node levels. When issuing a response, the CPE MUST match each Object in the Parameter Path Name at each node level with the same Instance Identifier type (Instance Number or Instance Alias) in the ACS request. All permutations (in any order) present in the following table are valid and MUST be supported:

Path Type	Message	Path Name Example
Uniform Instance Number Identifier	Request	TopGroup.Lev1Obj.1.Lev2Obj.1.
	Response	TopGroup.Lev1Obj.1.Lev2Obj.1.Parameter
Uniform Instance Alias Identifier	Request	TopGroup.Lev1Obj.[a].Lev2Obj.[b].
	Response	TopGroup.Lev1Obj.[a].Lev2Obj.[b].Parameter
Mixed Instance Identifier	Request	TopGroup.Lev1Obj.1.Lev2Obj.[b].
	Response	TopGroup.Lev1Obj.1.Lev2Obj.[b].Parameter

7. If the CPE has to issue a response that contains Object instances in the Parameter Path Name with node levels below the Path Node that was received in the ACS request, it MUST use the ManagementServer.InstanceMode Parameter to choose how to provide the Path Name in the response:
    - o If the ManagementServer.InstanceMode Parameter is set to InstanceNumber, all the Objects below the received Partial Path Name MUST be returned using Instance Number identifiers only. All the permutations (in any order) present in the following table are valid and MUST be supported:

Path Type	Message	Path Name Example
Uniform Instance Number identifier	Request	TopGroup.Lev1Obj.1.Lev2Obj.1.
	Response	TopGroup.Lev1Obj.1.Lev2Obj.1.Lev3Obj.1.Parameter
Uniform Instance Alias identifier	Request	TopGroup.Lev1Obj.[a].Lev2Obj.[b].
	Response	TopGroup.Lev1Obj.[a].Lev2Obj.[b]. Lev3Obj.1.Parameter
Mixed Instance identifier	Request	TopGroup.Lev1Obj.1.Lev2Obj.[b].
	Response	TopGroup.Lev1Obj.1.Lev2Obj.[b]. Lev3Obj.1.Parameter

    - o If the ManagementServer.InstanceMode Parameter is set to InstanceAlias, all the Objects located below the received Partial Path Name MUST contain Instance Alias identifiers where such identifiers exist. All the permutations (in any order) present in the following table are valid and MUST be supported:

Path Type	Message	Path Name Example
Uniform Instance Number identifier	Request	TopGroup.Lev1Obj.1.Lev2Obj.1.
	Response	TopGroup.Lev1Obj.1.Lev2Obj.1.Lev3Obj.[c].Parameter
Uniform Instance Alias identifier	Request	TopGroup.Lev1Obj.[a].Lev2Obj.[b].
	Response	TopGroup.Lev1Obj.[a].Lev2Obj.[b].Lev3Obj.[c].Parameter
Mixed Instance identifier	Request	TopGroup.Lev1Obj.1.Lev2Obj.[b].
	Response	TopGroup.Lev1Obj.1.Lev2Obj.[b]. Lev3Obj.[c].Parameter

8. The ManagementServer.InstanceMode Parameter affects the ParameterList argument of the CPE's Inform RPC, by how the CPE returns Parameter Path Names.

- If the ManagementServer.InstanceMode Parameter is set to InstanceNumber, then all the Objects in Parameter Path Names MUST use Instance Number identifiers only. For example:

Path Type	Path Name Example
Uniform Instance Number identifier	TopGroup.Lev1Obj.1.Lev2Obj.1.Parameter

- If the ManagementServer.InstanceMode Parameter is set to InstanceAlias, then all the Objects in Parameter Path Names MUST use Instance Alias identifiers where such identifiers exist. For example:

Path Type	Path Name Example
Uniform Instance Alias identifier	TopGroup.Lev1Obj.[a].Lev2Obj.[b].

9. The ManagementServer.InstanceMode Parameter also affects how the CPE returns the Parameter[15] values that are Path Names or lists of Path Names.
   - If the ManagementServer.InstanceMode Parameter is set to InstanceNumber, then all the Parameter values that are Path Names or lists of Path Names MUST be returned using Instance Number identifiers only. For example:

Path Type	Path Name Example
Uniform Instance Number identifier	TopGroup.Lev1Obj.1.Lev2Obj.1.

   - If the ManagementServer.InstanceMode Parameter is set to InstanceAlias, then all the Parameter values that are Path Names or lists of Path Names MUST be returned as follows:
     - For the Parameter values that were not generated by the ACS via a SetParameterValues or AddObject, using Instance Alias identifiers where such identifiers exist. For example:

Path Type	Path Name Example
Uniform Instance Alias identifier	TopGroup.Lev1Obj.[cpe-1].Lev2Obj.[cpe-2].

     - For the Parameter values that were generated by the ACS via a SetParameterValues or AddObject, using the same Instance Identifier types used when they were set. For example:

[15] This rule does not apply when the Parameter is a weak reference (Section 3.2.4/TR-106 [16]). In this case, the stored value is always returned.

Path Type	Action	Parameter Value Path Name Example
Uniform Instance Number identifier	Set	TopGroup.Lev1Obj.1.Lev2Obj.1
	Returned	TopGroup.Lev1Obj.1.Lev2Obj.1
Uniform Instance Alias identifier	Set	TopGroup.Lev1Obj.[a].Lev2Obj.[b]
	Returned	TopGroup.Lev1Obj.[a].Lev2Obj.[b]
Mixed Instance identifier	Set	TopGroup.Lev1Obj.1.Lev2Obj.[b]
	Returned	TopGroup.Lev1Obj.1.Lev2Obj.[b]

10. The CPE MUST change its ManagementServer.InstanceMode Parameter to its factory default value upon any event that requires the CPE to issue a BOOTSTRAP event.

3.6.2 Object Instance Wildcards Requirements

The use of object instance wildcards in parameter names is OPTIONAL to the ACS and the CPE. It makes use of the instance wildcards defined in A.2.4.

An ACS that supports object instance wildcards MUST fully comply with all the following requirements:

An ACS MUST only use object instance wildcards with a CPE that has indicated its support by setting the "ManagementServer.InstanceWildcardsSupported" Parameter (set to true).

Before using the object instance wildcards the first time after a "0 BOOTSTRAP" event, the ACS MUST query the CPE for the value of this parameter. If the ACS does not store the result of this parameter it MUST query the CPE again before using wild cards in a session.

A CPE that supports the object instance wildcards MUST fully comply with all the following requirements:

A CPE MUST support instance wildcards in parameter names as alternative methods to address Multi-Instance Objects in addition to Instance Number identifiers.

The CPE MUST indicates its support by implementing the "ManagementServer.-InstanceWildcardsSupported" Parameter and setting it to true.

3.6.3 Reference Parameter Requirements

The following requirements relate to reference types and the associated CPE behavior:

- A CPE MUST reject an attempt to set a strong reference parameter if the new value does not reference an existing parameter or object.
- A CPE MUST NOT reject an attempt to set a weak reference parameter because the new value does not reference an existing parameter or object.
- A CPE MUST change the value of a non-list-valued strong reference parameter to a null reference when a referenced parameter or object is deleted.
- A CPE MUST remove the corresponding list item from a list-valued strong reference parameter when a referenced parameter or object is deleted.
- A CPE MUST NOT change the value of a weak reference parameter when a referenced parameter or object is deleted.

The following requirements apply when a reference parameter contains Instance Aliases (as defined in Section A.2.2.2):

- Strong reference parameters refer to actual instances. When the alias of an instance is changed and there are strong reference parameters referencing a Parameter or Object whose path includes that instance, the CPE MUST keep those strong reference parameters referencing the same actual Parameter or Object after the alias change.
- Weak reference parameter values are stored as Path Names. A weak reference parameter therefore always references whichever Parameter or Object (if any) is currently referenced by the stored Path Name. This implies that, if the stored Path Name includes aliases, a change to any of those aliases will cause the weak reference parameter to reference a different Parameter or Object (or to reference nothing).

3.7 Session Procedures

All Sessions MUST begin with an Inform message from the CPE contained in the initial HTTP POST. This serves to initiate the set of transactions and communicate the limitations of the CPE with regard to message encoding. An Inform message MUST NOT occur more than once during a Session (this limitation does not apply to the potential need to retransmit an Inform request due to an HTTP "401 Unauthorized" status code received as part of the HTTP authentication process, or due to an HTTP 3xx status code received as part of an HTTP redirect).

The Session ceases when both the ACS and CPE have no more requests to send and no responses remain due from either the ACS or the CPE. At such time, the CPE MUST close the connection.

No more than one Session between a CWMP Endpoint and its associated ACS can exist at a time.

Note – a Session is intended to persist only as long as there are messages to be transferred in either direction. A Session and its associated TCP connection are not intended to be held open after a specific exchange of information completes.

3.7.1 CPE Operation

3.7.1.1 Session Initiation

The CPE will initiate a Session to the ACS as a result of the conditions listed in Section 3.2.1. Once the connection to the ACS is successfully established, the CPE initiates a Session by sending an initial Inform request to the ACS. This indicates to the ACS the current status of the CPE and that the CPE is ready to accept requests from the ACS.

A CPE that supports CWMP version 1.4 (or later) will include in every Inform request (in a SOAP SupportedCWMPVersions header) a comma-separated list of all of the CWMP version numbers that it supports.

- If a CMWP version 1.4 (or newer) CPE receives a UseCWMPVersion header in the InformResponse containing a version that the CPE sent in the SupportedCWMPVersions header, then the CPE MUST use the CWMP version returned in UseCWMPVersion.
- If a CWMP version 1.4 (or newer) CPE receives a UseCWMPVersion header in the InformResponse containing a version that the CPE did not send in the SupportedCWMPVersions header, then the CPE MUST abort session initiation by closing the TCP connection.
- If a CPE supporting any CWMP version does not receive a UseCWMPVersion, the CPE MUST infer the CWMP version from the CWMP namespace in the InformResponse returned from the ACS. Table 6 gives the mapping between CWMP namespace and CWMP version for the CPE.

Table 6 – Inferring ACS CWMP Version 1.0-1.3 from CWMP Namespace

CWMP Namespace	CWMP Version
urn:dslforum-org:cwmp-1-0	1.0
urn:dslforum-org:cwmp-1-1	1.1
urn:dslforum-org:cwmp-1-2	1.2 (there is no way for the CPE to infer 1.3)

For the remainder of the CWMP session, the CPE MUST NOT use any feature that is incompatible with the CWMP version chosen by the CPE.

The CPE MUST consider the Session to have been successfully initiated if and only if it receives a successful InformResponse.

From the time a Session is initiated until the Session is terminated, the CPE MUST ensure the transactional integrity of all Parameters accessible via the CPE WAN Management Protocol. During the course of a Session, all configurable Parameters of the CPE MUST appear to the ACS as a consistent set modified only by the ACS. Throughout the Session the CPE MUST shield the ACS from seeing any updates to the Parameters performed by other entities. This includes the values of configurable Parameters as well as presence or absence of configurable Parameters and Objects. The means by which the CPE achieves this transactional integrity is a local matter.

The CPE MUST take any necessary steps to ensure transactional integrity of the Session. For example, it might be necessary, in exceptional cases, for the CPE to terminate a LAN-side management session in order to meet CWMP Session establishment requirements.

3.7.1.2 Incoming Requests

While in a Session (after the Session was successfully initiated, but before the Session termination criteria described in 3.7.1.4 have been met), on reception of a SOAP request from the ACS, the CPE MUST respond to that request in the next HTTP POST that it sends to the ACS.

3.7.1.3 Outgoing Requests

While in a Session (after the Session was successfully initiated, but before the Session termination criteria described in 3.7.1.4 have been met), if the CPE has one or more requests to send to the ACS, the CPE MUST send one of these requests in the next HTTP POST if and only if all of the following conditions are met:

1) The most recently received HTTP response from the ACS did not contain a SOAP request.
2) The ACS has indicated that HoldRequests is false (see Section 3.4.7). This condition is met if and only if the most recently received HTTP response from the ACS contained one of the following:
   - A SOAP envelope with the HoldRequests header set to a value of false.
   - A SOAP envelope with no HoldRequests header.
   - No SOAP envelope (an empty HTTP response).

   Note – the HoldRequests SOAP Header element is DEPRECATED (see Section 3.4.7), so the ACS is not expected to send it. However, the ACS might send it, so the CPE still needs to support it.
3) At any prior time during the current Session, the CPE has not sent an empty HTTP POST at a time that the ACS had indicated that HoldRequests is false (as described above).

If the CPE has more than one request pending when the above criteria are met, the choice of which request to send is at the discretion of the CPE unless otherwise specified.

While in a Session, if any of the above conditions are not met or if the CPE has no requests to send to the ACS, and if the most recent HTTP response from the ACS did not contain a SOAP request, the CPE MUST send an empty HTTP POST.

Once the CPE has sent an empty HTTP POST when the most recent HoldRequests was false (see Section 3.4.7), the CPE MUST NOT send any further requests for the remainder of the Session. In this case, if the CPE has additional requests to send to the ACS, the CPE MUST wait until a subsequent Session to send these requests.

Table 7 summarizes what the CPE MUST send to the ACS as long as the Session is in progress (after the Session was successfully initiated, but before the Session termination criteria described in 3.7.1.4 have been met).

Table 7 – CPE Message Transmission Constraints

	HoldRequests	**ACS request outstanding**	**No ACS request outstanding**
CPE requests pending[16]	**false**	Response	Request
	true	Response	Empty HTTP POST
No CPE requests pending	**-**	Response	Empty HTTP POST

3.7.1.4 Session Termination

The CPE MUST terminate the Session when <u>all</u> of the following conditions are met:

1) The ACS has no further requests to send the CPE. The CPE concludes this if and only if the most recent HTTP response from the ACS was empty.
2) The CPE has no further requests to send to the ACS and the CPE has issued an empty HTTP POST to the ACS while HoldRequests is false (which indicates to the ACS that the CPE has no further requests for the remainder of the Session). As defined in Table 7, if this condition has not been met but the CPE has no further requests or responses, it MUST send an empty HTTP POST, which will then fulfill this condition.

   Note – the HoldRequests SOAP Header element is DEPRECATED (see Section 3.5), so the ACS is not expected to send it. However, the ACS might send it, so the CPE still needs to support it.

3) The CPE has received all outstanding response messages from the ACS.
4) The CPE has sent all outstanding response messages to the ACS resulting from prior requests.

The CPE MUST also consider a Session unsuccessfully terminated if it has received no HTTP response from an ACS for a locally determined time period of not less than 30 seconds. If the CPE fails to receive an HTTP response, the CPE MUST NOT attempt to retransmit the corresponding HTTP request as part of the same Session.

If the CPE receives a SOAP-layer fault in response to an Inform request with a fault code other than “Retry request” (fault code 8005), the CPE MUST consider the Session to have terminated unsuccessfully.

If the CPE receives an HTTP response from the ACS for which the XML is not well-formed, for which the SOAP structure is deemed invalid, that contains a SOAP fault that is not in the form specified in Section 3.5, or for which the CPE deems that the protocol has been violated, the CPE MUST consider the Session to have terminated unsuccessfully.

[16] The CPE can have requests pending only if the CPE has not already sent an empty HTTP POST when the most recent HoldRequests was false. Otherwise, the CPE is considered to have no requests pending.

If the CPE receives an HTTP response from the ACS with a fault status code (a 4xx or 5xx status code) that is not otherwise handled by the CPE, the CPE MUST consider the Session to have terminated unsuccessfully. Note that while the CPE would accept an HTTP response with a "401 Unauthorized" status code as part of the normal authentication process, when the CPE subsequently attempts to authenticate, if the resulting HTTP response contains a "401 Unauthorized" status code, the CPE MUST consider the Session to have terminated unsuccessfully.

If the above conditions are not met, the CPE MUST continue the Session.

If the CPE receives a SOAP-layer fault response as defined in Section 3.5 with a fault code other than "Retry request" (fault code 8005) in response to any method other than Inform, the CPE MUST continue with the remainder of the Session. That is, a fault response of this type MUST NOT cause the Session to unsuccessfully terminate.

Note – in a fault condition, it is entirely at the discretion of the ACS whether its fault response is a SOAP-layer fault, which would cause the Session to continue, or an HTTP-layer fault, which would cause the Session to terminate unsuccessfully.

If one or more messages exchanged during a Session results in the CPE needing to reboot to complete the requested operation, the CPE MUST wait until after the Session has cleanly terminated based on the above criteria before performing the reboot.

If the Session terminates unexpectedly, the CPE MUST retry the Session as specified in Section 3.2.1.1. The CPE MAY place locally specified limits on the number of times it attempts to reestablish a Session in this case.

3.7.1.5 Events

An event is an indication that something of interest has happened that requires the CPE to notify the ACS via an Inform request defined in Section A.3.3.1. The CPE MUST attempt to deliver every event at least once. If the CPE is not currently in a Session with the ACS, it MUST attempt to deliver events immediately; otherwise, it MUST attempt to deliver them after the current Session terminates[17]. The CPE MUST receive confirmation from the ACS for it to consider an event successfully delivered. Once the CPE has delivered an event successfully, the CPE MUST NOT send the same event again. On the other hand, the ACS MUST be prepared to receive the same event more than once because the ACS might have sent a response the CPE never receives. Many types of events (e.g., PERIODIC, VALUE CHANGE) can legally appear in subsequent Sessions even when successfully delivered in the earlier Session. In such cases, an event in the later Session indicates the reoccurrence of an event of the same type rather than an attempt to retry an event delivery failure.

For every type of event there is a policy that dictates if and when the CPE MUST retry event delivery if a previous delivery attempt failed. When event delivery is retried it MUST be in the immediately following Session; events whose delivery fails in one Session cannot be omitted in the following Session and then later redelivered.

[17] VALUE CHANGE events resulting from a passive notification are an exception to this rule because these events (and associated parameters) are supposed to be sent as part of an Inform the next time a session is established as a result of some other event (eg: active notification, connection request, etc).

For most events, delivery is confirmed when the CPE receives a successful InformResponse. Six standard event types (KICKED[18], TRANSFER COMPLETE, AUTONOMOUS TRANSFER COMPLETE, REQUEST DOWNLOAD, DU STATE CHANGE COMPLETE, and AUTONOMOUS DU STATE CHANGE COMPLETE) indicate that one or more methods (Kicked [Section A.4.2.1], TransferComplete [Section A.3.3.2], AutonomousTransferComplete [Section A.3.3.3], RequestDownload [Section A.4.2.2], DUStateChangeComplete [Section A.4.2.3], AutonomousDUStateChange-Complete [Section A.4.2.4] respectively) will be called later in the Session, and it is the successful response to these methods that indicates event delivery. The CPE MAY also send vendor-specific events (using the syntax specified in Table 8), in which case successful delivery, retry, and discard policy is subject to vendor definition.

If no new events occur while the CPE has some events to redeliver, the CPE MUST attempt to redeliver them according to the schedule defined by the session retry policy in Section 3.2.1.1.

Below is a table of event types, their codes in an Inform request, their cumulative behavior, the responses the CPE MUST receive to consider them successfully delivered, and the policy for retrying and/or discarding them if delivery is unsuccessful.

Table 8 – Event Types

Event Code	Cumulative Behavior	Explanation	ACS Response for Successful Delivery	Retry/Discard Policy
"0 BOOTSTRAP"	Single	Indicates that the Session was established due to first-time CPE installation or a change to the ACS URL. The specific conditions that MUST result in the BOOTSTRAP EventCode are: • First time connection of the CWMP Endpoint to the ACS from the factory. • First time connection of the CWMP Endpoint to the ACS after a factory reset. • First time connection of the CWMP Endpoint to the ACS after the ACS URL has been modified in any way. Note that as with all other EventCode values, the BOOTSTRAP EventCode MAY be included in the Event array along with other EventCode values. It would be expected, for example, that on the initial boot of the CPE from the factory, the CPE would include both the BOOTSTRAP and BOOT EventCodes.	InformResponse	The CPE MUST NOT ever discard an undelivered BOOTSTRAP event. All other undelivered events MUST be discarded on BOOTSTRAP.

[18] DEPRECATED due to the deprecation of Annex D, which is the Section that defined the usage of this Event.

Event Code	Cumulative Behavior	Explanation	ACS Response for Successful Delivery	Retry/Discard Policy
"1 BOOT"	Single	Indicates that the Session was established due to the CPE being powered up or reset. This includes initial system boot, as well as reboot due to any cause, including use of the Reboot method, but not waking up from standby.	InformResponse	The CPE MUST retry delivery until it reboots before discarding it.
"2 PERIODIC"	Single	Indicates that the Session was established on a periodic Inform interval.	InformResponse	The CPE MUST NOT ever discard an undelivered PERIODIC event (except on BOOTSTRAP).
"3 SCHEDULED"	Single	Indicates that the Session was established due to a ScheduleInform method call. This event code MUST only be used with the "M ScheduleInform" event code (see "M ScheduleInform", below).	InformResponse	The CPE MUST NOT ever discard an undelivered SCHEDULED event (except on BOOTSTRAP).
"4 VALUE CHANGE"	Single	Indicates that since the last successful Non-HEARTBEAT Inform (under the conditions defined in Section A.3.2.4), the value of one or more Parameters with Passive or Active notification enabled (including Parameters defined to require Forced Active Notification) has been modified (even if its value has changed back to the value it had at the time of the last successful Inform). If this EventCode is included in the Event array, all such modified Parameters MUST be included in the ParameterList in this Inform. If this event is ever discarded then the list of modified Parameters MUST be discarded at the same time.	InformResponse	The CPE MUST retry delivery until it reboots. It MUST discard the event on BOOTSTRAP.
"5 KICKED"[19] **(DEPRECATED)**	Single	Indicates that the Session was established for the purpose of web identity management (see Annex D) and that a Kicked method (see Section A.4.2.1) will be called one or more times during this Session.	KickedResponse	The CPE MAY retry delivery at its discretion.
"6 CONNECTION REQUEST"	Single	Indicates that the Session was established due to a Connection Request from the ACS as described in Section 3.2.	InformResponse	The CPE MUST NOT retry delivery.

[19] DEPRECATED due to the deprecation of Annex D, which is the Section that defined the usage of this Event.

Event Code	Cumulative Behavior	Explanation	ACS Response for Successful Delivery	Retry/Discard Policy
“7 TRANSFER COMPLETE”	Single	Indicates that the Session was established to indicate the completion of a previously requested download or upload (either successful or unsuccessful) and that the TransferComplete method will be called one or more times during this Session. This event code MUST only be used with the “M Download”, “M ScheduleDownload” and/or “M Upload” event codes (see “M Download”, “M ScheduleDownload” and “M Upload”, below).	TransferCompleteResponse	The CPE MUST NOT ever discard an undelivered TRANSFER COMPLETE event (except on BOOTSTRAP).
"8 DIAGNOSTICS COMPLETE"	Single	Used when reestablishing a connection to the ACS after completing one or more diagnostic test initiated by the ACS. This event MUST NOT be sent if the test is canceled for any reason.	InformResponse	The CPE MUST retry delivery until it reboots before discarding it.
“9 REQUEST DOWNLOAD”	Single	Indicates that the Session was established for the CPE to call the RequestDownload method (see Section A.4.2.2) one or more times.	RequestDownloadResponse	The CPE MAY retry delivery at its discretion.
“10 AUTONOMOUS TRANSFER COMPLETE”	Single	Indicates that the Session was established to indicate the completion of a download or upload that was not specifically requested by the ACS (either successful or unsuccessful) and that the Autonomous-TransferComplete method will be called one or more times during this Session.	AutonomousTransfer-CompleteResponse	The CPE MUST NOT ever discard an undelivered AUTONOMOUS TRANSFER COMPLETE event (except on BOOTSTRAP).
“11 DU STATE CHANGE COMPLETE”	Single	Indicates that the Session was established to indicate the completion of a previously requested DU state change, either successful or unsuccessful, and that the DUStateChangeComplete method will be called during this Session. This method MUST only be used with the “M ChangeDUState” event code (see “M ChangeDUState”, below).	DUStateChangeComplete-Response	The CPE MUST NOT ever discard an undelivered DU STATE CHANGE COMPLETE event (except on BOOTSTRAP).
“12 AUTONOMOUS DU STATE CHANGE COMPLETE”	Single	Indicates that the Session was established to indicate the completion of a DU state change not specifically requested by a ChangeDUState RPC (either successful or unsuccessful) and that the Autonomous-DUStateChangeComplete method will be called during this Session.	AutonomousDUState-ChangeCompleteResponse	The CPE MUST NOT ever discard an undelivered AUTONOMOUS DU STATE CHANGE COMPLETE event (except on BOOTSTRAP).

Event Code	Cumulative Behavior	Explanation	ACS Response for Successful Delivery	Retry/Discard Policy
“13 WAKEUP”	Single	Indicates that the Session was established due to the CPE waking up from standby under the conditions specified in L.2. A device that implements the WAKEUP event code MUST implement the StandbyPolicy:1 profile as defined in tr-157-1-8.xml [41]	InformResponse	The CPE MUST retry delivery until it reboots. It MUST discard the event on BOOTSTRAP.
“14 HEARTBEAT”	Single	Indicates that the Session was established when conditions of the CPE’s Heartbeat policy has been fulfilled. A device that implements the HEARTBEAT event code MUST implement the HeartbeatPolicy:1 profile as defined in tr-181i2 [35].	InformResponse	The CPE MUST retry delivery until it reboots before discarding it.
“M Reboot”	Multiple	The CPE rebooted upon request from the ACS through the use of the Reboot RPC. Overlaps with one of the causes that can generate a “1 BOOT” event code.	InformResponse	The CPE MUST NOT ever discard an undelivered “M Reboot” event (except on BOOTSTRAP).
“M ScheduleInform”	Multiple	The ACS requested a scheduled Inform.	InformResponse	The CPE MUST NOT ever discard an undelivered “M ScheduleInform” event (except on BOOTSTRAP).
“M Download”	Multiple	A content download previously requested by the ACS using the Download method (see Section A.3.2.8) has finished. Overlaps with “7 TRANSFER COMPLETE”.	TransferCompleteResponse	The CPE MUST NOT ever discard an undelivered “M Download” event (except on BOOTSTRAP).
“M ScheduleDownload”	Multiple	A content download previously requested by the ACS using the ScheduleDownload method (see Section A.4.1.8) has finished. Overlaps with “7 TRANSFER COMPLETE”.	TransferCompleteResponse	The CPE MUST NOT ever discard an undelivered “M ScheduleDownload” event (except on BOOTSTRAP).
“M Upload”	Multiple	A content upload previously requested by the ACS using the Upload method (see Section A.4.1.5) has finished. Overlaps with “7 TRANSFER COMPLETE”.	TransferCompleteResponse	The CPE MUST NOT ever discard an undelivered “M Upload” event (except on BOOTSTRAP).
“M ChangeDUState”	Multiple	A DU state change previously requested by the ACS using the ChangeDUState method (see Section A.4.1.10) has finished. Overlaps with “11 DU STATE CHANGE COMPLETE”.	DUStateChangeComplete-Response	The CPE MUST NOT ever discard an undelivered “M ChangeDUState” event (except on BOOTSTRAP).

Event Code	Cumulative Behavior	Explanation	ACS Response for Successful Delivery	Retry/Discard Policy
"M " <vendor-specific method>	Not specified	The action requested by a vendor-specific method is complete. The action taken by the CPE and response by the ACS is vendor-specific. A vendor-specific method name MUST be in the form specified in Section A.3.1.1. For example: "M X_012345_MyMethod"	Not specified	Not specified
"X "<VENDOR> " " <event>	Not specified	Vendor-specific event. The VENDOR after the "X" and space is a unique vendor identifier, which MAY be either an OUI or a domain name. The OUI or domain name used for a given vendor-specific event MUST be one that is assigned to the organization that defined this method (which is not necessarily the same as the vendor of the CPE or ACS). An OUI is an organizationally unique identifier as defined in [13], which MUST be formatted as a 6 hexadecimal-digit OUI (organizationally unique identifier), with all upper-case letters and any leading zeros included. A domain name MUST be upper case with each dot (".") replaced with a hyphen or underscore. For example: "X 012345 MyEvent" "X ACME_COM MyEvent"	Not specified	Not specified

The Cumulative Behavior column of the above table distinguishes between event types that are not cumulative ("Single") and those that are cumulative ("Multiple"). For example, if the CPE reboots while the previous "1 BOOT" event has not yet been delivered, it makes no sense for the next Inform to contain two "1 BOOT" Event array entries. In contrast, if a download completes while the previous "M Download" event has not yet been delivered, the next Inform would contain two "M Download" Event array entries because each relates to a different ACS request. The "Single" and "Multiple" cumulative behaviors are defined as follows:

- If an event with "Single" cumulative behavior occurs, the list of events in the next Inform MUST contain only one instance of this EventCode, regardless of whether there are any undelivered events of the same type.
- If an event with "Multiple" cumulative behavior occurs, the new EventCode MUST be included in the list of events, independent of any undelivered events of the same type, and this MUST NOT affect any such undelivered events.

When one or more events are directly related to the same root cause, then all such events MUST be included in the Event array. Below are examples of such cases (this list is *not* exhaustive):

- Reboot caused by the Reboot RPC method. In this case the Inform MUST include at least the following EventCode values:

```
"1 BOOT"
"M Reboot"
```

- TransferComplete sent in a new Session due to a prior Download request, where there is no reboot associated with the completion of the transfer:

```
"7 TRANSFER COMPLETE"
"M Download"
```

- One or more Parameter values for which Passive notification has been set have changed since the most recent Inform, and a periodic Inform occurs (in this case, the events MUST be included in the same Inform because for Passive notifications, the Inform in which the “4 VALUE CHANGE” event would occur would have to result from some other cause—in this example, a periodic inform):

```
"2 PERIODIC"
"4 VALUE CHANGE"
```

For events that are due to unrelated causes, if they occur simultaneously, the CPE SHOULD include all such events in the same Inform message, but MAY send separate Inform messages for each such event. An example of unrelated events is:

```
"2 PERIODIC"
"7 TRANSFER COMPLETE"
```

3.7.1.6 Method Retry Behavior

If in response to a request from the CPE the CPE receives a “Retry request” response (fault code 8005) from the ACS, the CPE MUST resend the identical request in the next HTTP POST within the current Session. The number of retries within the current session MUST be limited to 3 and the CPE MUST consider the session failed and terminate it if the session retry limit has been exceeded. This behavior applies to all ACS methods (including Inform).

If instead the CPE receives a fault response with any fault code other than 8005 in response to any method other than Inform, the CPE MUST proceed with the Session, and MUST NOT attempt to retry the method (such a response in the case of Inform will terminate the Session, as described in Section 3.7.1.4).

3.7.2 ACS Operation

3.7.2.1 Session Initiation

Upon receiving the initial Inform request from the CPE, if the ACS wishes to allow the initiation of the Session, it MUST respond with an InformResponse.

An ACS that supports CWMP version 1.4 (or later) MUST understand the Supported-CWMPVersions header.

- If a CWMP version 1.4 (or newer) ACS receives the SupportedCWMPVersions SOAP header, the ACS MUST choose the appropriate version for this CWMP session from the list of versions in the SupportedCWMPVersions header, and return that version to the CPE in a SOAP UseCWMPVersion header in the InformResponse.
- If a CWMP version 1.4 (or newer) ACS receives the SupportedCWMPVersions SOAP header, and the ACS does not support any of the versions in the list of versions in the SupportedCWMPVersions header, the ACS MUST abort session initiation by returning an 8006 SOAP-layer fault.
- If a CWMP version 1.4 (or newer) ACS does not receive the SupportedCWMP-Versions SOAP header, then the ACS MUST infer the CWMP version (which will be between 1.0 and 1.3 inclusive) from the CWMP namespace in the SOAP envelope of the Inform request. Table 9 gives the mapping between CWMP namespace and CWMP version for the ACS. In this case, the ACS MUST NOT send the UseCWMPVersion SOAP header in the InformResponse.
- If a CWMP version 1.3 (or older) ACS receives the SupportedCWMPVersions SOAP header, the ACS will most likely ignore the SOAP header (since it does not understand it and mustUnderstand is set to “false”). The ACS will most likely infer the CWMP version (which will be between 1.0 and 1.3 inclusive) from the CWMP namespace in the SOAP envelope of the Inform request. Table 9 gives the mapping between CWMP namespace and CWMP version for the ACS. In this case, the ACS will not send the UseCWMPVersion SOAP header in the InformResponse.

Table 9 – Inferring CPE CWMP Version 1.0-1.3 from CWMP Namespace

CWMP Namespace	CWMP Version
urn:dslforum-org:cwmp-1-0	1.0
urn:dslforum-org:cwmp-1-1	1.1
urn:dslforum-org:cwmp-1-2	1.2 if the SOAP SessionTimeout header is absent, otherwise 1.3

For the rest of the CWMP session, the ACS MUST NOT use any feature that is incompatible with the CWMP version chosen by the ACS.

Note – an ACS that supports only v1.0 of the CPE WAN Management Protocol will expect the initial Inform request from the CPE to use the v1.0 namespace identifier “urn:dslforum-org:cwmp-1-0”, and to contain only event types that were defined in v1.0 of the protocol. The behavior of such an ACS when it receives an initial Inform from a CPE that supports v1.1 (or later) is not possible to predict. The ACS might fail to notice that the CPE supports a later version, in which case it will respond with an InformResponse; it might return a SOAP-layer fault; or it might return an HTTP-layer fault. If it returns a fault, the CPE will need to decide whether or not to revert to v1.0 of the protocol when retrying the failed Session.

The ACS MUST ignore any event types that it does not recognize.

3.7.2.2 Incoming Requests

While in a Session (after the Session was successfully initiated, but before the Session termination criteria described in 3.7.2.4 have been met), on reception of a SOAP request from the CPE, the ACS MUST respond to that request in the next HTTP response sent to the CPE.

Note – the HoldRequests SOAP Header element is DEPRECATED (see Section 3.4.7), so the ACS SHOULD NOT send it.

3.7.2.3 Outgoing Requests

While in a Session (after the Session was successfully initiated, but before the Session termination criteria described in 3.7.2.4 have been met), if the ACS has one or more requests to send to the CPE and the most recent HTTP POST from the CPE did not contain a SOAP request, the ACS MUST send one of these requests in the next HTTP response.

Otherwise, while in a Session, if the ACS has no requests to send to the CPE and the most recent HTTP POST from the CPE did not contain a SOAP request, the ACS MUST send an empty HTTP response.

Table 10 summarizes what the ACS MUST send to the CPE as long as the Session is in progress (after the Session was successfully initiated, but before the Session termination criteria described in 3.7.2.4 have been met).

Table 10 – ACS Message Transmission Constraints

	CPE request outstanding	**No CPE request outstanding**
ACS requests pending	Response	Request
No ACS requests pending	Response	Empty HTTP response

3.7.2.4 Session Termination

Since the CPE is driving the HTTP connection to the ACS, only the CPE is responsible for connection initiation and teardown.

The ACS MUST consider the Session terminated when <u>all</u> of the following conditions are met:

1) The CPE has no further requests to send the ACS. The ACS concludes this if and only if it has received an empty HTTP POST from the CPE while HoldRequests is false.

   Note – the HoldRequests SOAP Header element is DEPRECATED (see Section 3.5), so the ACS SHOULD NOT send it.

2) The ACS has no further requests to send the CPE and the most recent HTTP response the ACS sent to the CPE was empty (which indicates to the CPE that the ACS has no further requests).
3) The ACS has sent all outstanding response messages to the CPE resulting from prior requests.
4) The ACS has received all outstanding response messages from the CPE.

If all of the above criteria have been met before the ACS has sent its final HTTP response, the final HTTP response from the ACS MUST be empty.

If the above criteria have not all been met, but the ACS has not received an HTTP POST from a given CPE within a locally defined timeout of not less than 30 seconds, it MAY consider the Session terminated. In this case, the ACS MAY attempt to reestablish a Session by performing a Connection Request (see Section 3.2.2).

If the ACS receives an HTTP POST from the CPE for which the XML is not well-formed, for which the SOAP structure is deemed invalid, or that contains a SOAP fault that is not in the form specified in Section 3.5, the ACS MUST respond to the CPE with an HTTP 400 status code (Bad Request), and MUST consider the Session to have terminated unsuccessfully. This fault response MUST NOT contain any SOAP content, but MAY contain human-readable text that further explains the nature of the fault.

If the ACS receives a request associated with a Session that it considers expired, or if the ACS determines that some other protocol violation has occurred, or for other reasons at the discretion of the ACS[20], the ACS MAY cause a Session to terminate unsuccessfully by responding to the CPE with an HTTP 400 status code (Bad Request). This HTTP response MUST NOT contain any SOAP content, but MAY contain human readable-text that further explains the nature of the fault.

If the ACS receives a SOAP fault response from the CPE, as defined in Section 3.5, the ACS MUST interpret any unrecognized fault code between 9000 and 9799 (inclusive) the same as 9001 (Request denied), and MAY choose among the following actions:

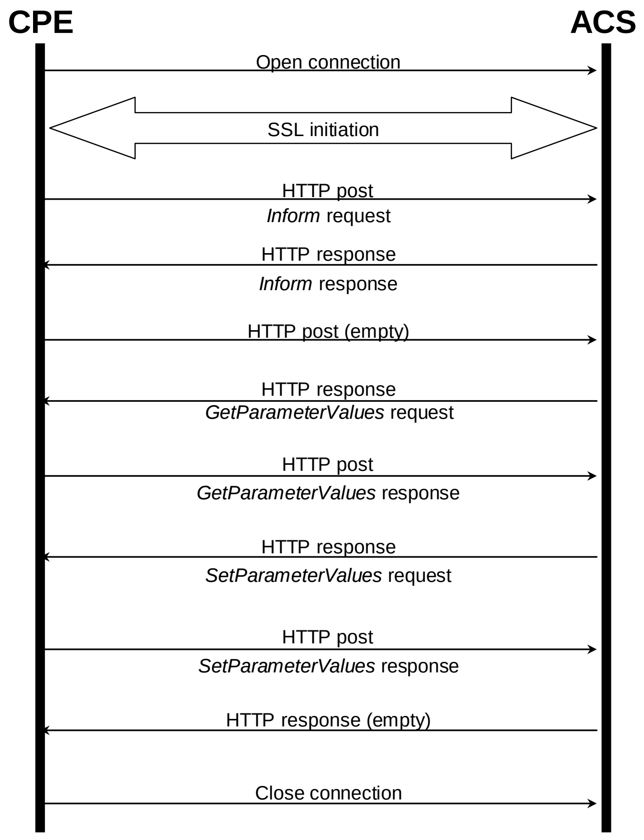
- The ACS MAY force the unsuccessful termination of the Session. To do this, the ACS MUST respond to the CPE with an HTTP 400 status code (Bad Request). This HTTP response MUST NOT contain any SOAP content, but MAY contain human readable-text that further explains the nature of the fault. This will result in the CPE retrying the Session.
- The ACS MAY attempt to terminate the Session successfully, in which case the CPE will not attempt to retry the Session. To do this, the ACS would send no more requests to the CPE, and would follow the rules defined above to determine when the Session terminates.
- The ACS MAY continue with the Session, sending additional requests to the CPE.

[20] With the exception that reception of a SOAP request to invoke an unsupported RPC method MUST result in a SOAP-layer fault response with a fault code indicating “Method not supported” (fault code 8000).

3.7.3 Transaction Examples

In the example shown in Figure 3, the ACS first reads a set of Parameter values, and based on the result, sets some Parameter values.

Figure 3 – Session Example



3.7.4 CWMP Version Negotiation

The following table illustrates the CWMP versions that will be negotiated in all possible cases.

Table 11 – CWMP Version Negotiation

	1.0 CPE	**1.1-1.3 CPE**	**1.4 (or later) CPE**
1.0 ACS	1. CPE sends 1.0 namespace 2. ACS sends a 1.0 namespace response that does not contain UseCWMPVersion header element 3. CPE chooses CWMP version 1.0 ACS chooses CWMP version 1.0	1. CPE sends 1.1-1.3 namespace 2. ACS sends a fault or a 1.0 namespace response 3. CPE chooses CWMP version 1.0 ACS chooses CWMP version 1.0	1. CPE sends SupportedCWMPVersions header element 2. ACS ignores SupportedCWMPVersion header element 3. ACS sends a fault or a v1.0 namespace response 4. CPE chooses CWMP version 1.0 ACS chooses CWMP version 1.0
1.1-1.3 ACS	1. CPE sends 1.0 namespace 2. ACS sends a 1.0 namespace response that does not contain UseCWMPVersion header element 3. CPE chooses CWMP version 1.0 ACS chooses CWMP version 1.0	1. CPE sends 1.1-1.3 namespace 2. ACS does not send UseCWMPVersion header element 3. CPE chooses `min`(CPE CWMP version, ACS CWMP version) ACS chooses `min`(CPE CWMP version, ACS CWMP version)	1. CPE sends SupportedCWMPVersions header element 2. ACS ignores SupportedCWMPVersions header element 3. ACS does not send UseCWMPVersion header element 4. CPE chooses ACS CWMP version ACS chooses ACS CWMP version
1.4 (or later) ACS	1. CPE sends 1.0 namespace 2. ACS sends a 1.0 namespace response that does not contain UseCWMPVersion header element 3. CPE chooses CWMP version 1.0 ACS chooses CWMP version 1.0	1. CPE sends 1.1-1.3 namespace 2. ACS does not send UseCWMPVersion header element 3. CPE chooses CPE CWMP version ACS chooses CPE CWMP version	1. CPE sends SupportedCWMPVersions header element 2. ACS uses unspecified logic to determine CWMP version from the supplied list 3. ACS replies with UseCWMPVersion header element 4. CPE chooses CWMP version specified by UseCWMPVersion ACS chooses CWMP version specified by UseCWMPVersion

Normative References

The following documents are referenced by this specification. Where the protocol defined in this specification depends on a referenced document, support for all required components of the referenced document is implied unless otherwise specified.

The following references are associated with document conventions or context for this specification, but are not associated with requirements of the CPE WAN Management Protocol itself.

[1] RFC 2119, *Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt

[2] TR-046, *Auto-Configuration Architecture & Framework*, Broadband Forum Technical Report

[3] TR-062, *Auto-Configuration for the Connection Between the DSL Broadband Network Termination (B-NT) and the Network using ATM*, Broadband Forum Technical Report

[4] TR-044, *Auto-Configuration for Basic Internet (IP-based) Services*, Broadband Forum Technical Report

The following references are associated with *required* components of the CPE WAN Management Protocol.

[5] RFC 1034, *Domain names – concepts and facilities,* http://www.ietf.org/rfc/rfc1034.txt

[6] RFC 7230, *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing, http://www.ietf.org/rfc/rfc7230.txt*

[7] RFC 7231, *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content,* http://www.ietf.org/rfc/rfc7231.txt

[8] RFC 7235, *Hypertext Transfer Protocol (HTTP/1.1): Authentication,* http://www.ietf.org/rfc/rfc7235.txt

[9] RFC 7616, *HTTP Digest Access Authentication*, http://www.ietf.org/rfc/rfc7616.txt

[10] RFC 7617, *The 'Basic' HTTP Authentication Scheme*, http://www.ietf.org/rfc/rfc7617.txt

[11] RFC 6265, *HTTP State Management Mechanism*, http://www.ietf.org/rfc/rfc6265.txt

[12] *Simple Object Access Protocol (SOAP) 1.1*, http://www.w3.org/TR/2000/NOTE-SOAP-20000508

[13] *Organizationally Unique Identifiers (OUIs)*, http://standards.ieee.org/faqs/OUI.html

[14] RFC 5246, *The Transport Layer Security (TLS) Protocol, Version 1.2*, http://www.ietf.org/rfc/rfc5246.txt

[15] RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*, http://www.ietf.org/rfc/rfc3986.txt

[16] TR-106, *Data Model Template for TR-069-Enabled Devices*, Broadband Forum Technical Report

The following references are associated with *optional* or *recommended* components of the CPE WAN Management Protocol.

[17] RFC 2132, DHCP Options and BOOTP Vendor Extensions, http://www.ietf.org/rfc/rfc2132.txt

[18] *XML-Signature Syntax and Processing*, http://www.w3.org/2000/09/xmldsig

[19] PKCS #7, *Cryptographic Message Syntax Standard*, http://www.rsasecurity.com/rsalabs/pkcs/pkcs-7/index.html or http://www.ietf.org/rfc/rfc2315.txt

[20] *Port Numbers*, http://www.iana.org/assignments/port-numbers

[21] *IANA Private Enterprise Numbers registry*, http://www.iana.org/assignments/enterprise-numbers

[22] RFC 2104, *HMAC: Keyed-Hashing for Message Authentication*, http://www.ietf.org/rfc/rfc2104.txt

[23] RFC 2131, *Dynamic Host Configuration Protocol*, http://www.ietf.org/rfc/rfc2131.txt

[24] RFC 3489, *STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)*, http://www.ietf.org/rfc/rfc3489.txt

[25] RFC 3925, *Vendor-Identifying Vendor Options for Dynamic Host Configuration Protocol version 4 (DHCPv4)*, http://www.ietf.org/rfc/rfc3925.txt

[26] *HTML 4.01 Specification*, http://www.w3.org/TR/html4

[27] TR-098, *Internet Gateway Device Data Model for TR-069*, Broadband Forum Technical Report (DEPRECATED)

[28] TR-104, *Provisioning Parameters for VoIP CPE*, Broadband Forum Technical Report

[29] TR-135, *Data Model for a TR-069 Enabled STB*, Broadband Forum Technical Report

[30] TR-140 Issue 1.1, *TR-069 Data Model for Storage Service Enabled Devices*, Broadband Forum Technical Report

[31] TR-143, *Enabling Network Throughput Performance Tests and Statistical Monitoring*, Broadband Forum Technical Report

[32] TR-157, *Component Objects for CWMP*, Broadband Forum Technical Report

[33] TR-196, *Femto Access Point Service Data Model*, Broadband Forum Technical Report

[34] TR-181 Issue 1, *Device Data Model for TR-069*, Broadband Forum Technical Report (DEPRECATED)

[35] TR-181 Issue 2, *Device Data Model for TR-069*, Broadband Forum Technical Report

[36] RFC 5389, *Session Traversal Utilities for NAT (STUN)*, http://www.ietf.org/rfc/rfc5389.txt

[37] RFC 4122, *A Universally Unique IDentifier (UUID) URN Namespace*, http://www.ietf.org/rfc/rfc4122.txt

[38] RFC 3315, *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*, http://www.ietf.org/rfc/rfc3315.txt

[39] IEEE 802a, *IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture - Amendment 1: Ethertypes for Prototype and Vendor-Specific Protocol Development.*

[40] RFC 6120, *Extensible Messaging and Presence Protocol (XMPP): Core*, http://www.ietf.org/rfc/rfc6120.txt

[41] tr-157-1-8.xml, *Component Objects for CWMP*, Broadband Forum, http://www.broadband-forum.org/cwmp/tr-157-1-8.xml

[42] UPnP IGD v1, *UPnP InternetGatewayDevice:1 Device Template*, UPnP Forum, http://upnp.org/specs/gw/UPnP-gw-InternetGatewayDevice-v1-Device.pdf

[43] UPnP IGD v2, *UPnP InternetGatewayDevice:2 Device Template*, UPnP Forum, http://upnp.org/specs/gw/UPnP-gw-InternetGatewayDevice-v2-Device.pdf

[44] UPnP WAN PPP v1, *UPnP WANPPPConnection:1 Service*, UPnP Forum, http://upnp.org/specs/gw/UPnP-gw-WANPPPConnection-v1-Service.pdf

[45] UPnP WAN IP v2, *UPnP WANIPConnection:2 Service*, UPnP Forum, http://upnp.org/specs/gw/UPnP-gw-WANIPConnection-v2-Service.pdf

[46] UPnP WAN IPv6 FW v1, *UPnP WANIPv6FirewallControl:1 Service*, UPnP Forum, http://upnp.org/specs/gw/UPnP-gw-WANIPv6FirewallControl-v1-Service.pdf

[47] RFC 6092, *Recommended Simple Security Capabilities in Customer Premise Equipment (CPE) for Providing Residential IPv6 Internet Services*, http://www.ietf.org/rfc/rfc6092.txt

[48] TR-124 Issue 3, *Functional Requirements for Broadband Residential Gateway Devices*, Broadband Forum Technical Report

[49] 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)*, https://tools.ietf.org/rfc/rfc6125.txt

[50] RFC 6066, *Transport Layer Security (TLS) Extensions: Extension Definitions*, https://tools.ietf.org/rfc/rfc6066.txt.

[51] RFC 5077, *Transport Layer Security (TLS) Session Resumption without Server-Side State*, https://tools.ietf.org/rfc/rfc5077.txt.

[52] RFC 4180, *Common Format and MIME Type for Comma-Separated Values (CSV) Files*, https://www.ietf.org/rfc/rfc4180.txt

[53] RFC 7159, *The JavaScript Object Notation (JSON) Data Interchange Format*, https://www.ietf.org/rfc/rfc7159.txt

[54] OWASP, *Cross-site Scripting*, https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

[55] OWASP, *Cross-Site Request Forgery*, https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

[56] RFC 7525, *Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*, http://www.ietf.org/rfc/rfc7525.txt

[57] RFC 7568, *Deprecating Secure Sockets Layer Version 3.0*, http://www.ietf.org/rfc/rfc7568.txt

[58] RFC 7395, *An Extensible Messaging and Presence Protocol (XMPP) Subprotocol for WebSocket*, http://www.ietf.org/rfc/rfc7395.txt

A list of currently valid Broadband Forum Technical Reports is published at www.broadband-forum.org.

Annex A. RPC Methods

A.1 Introduction

In the CPE WAN Management Protocol, a remote procedure call mechanism is used for bi-directional communication between a CPE device and an Auto-configuration Server (ACS). This Annex specifies the specific procedure calls (methods). This includes both methods initiated by an ACS and sent to a CPE, as well as methods initiated by a CPE and sent to an ACS.

This specification is intended to be independent of the syntax used to encode the defined RPC methods. The particular encoding syntax to be used in the context of the CPE WAN Management Protocol is defined in Section 3.4.7.

A.2 RPC Method Usage

A.2.1 Data Types

The RPC methods defined in this specification make use of a limited subset of the default SOAP data types [12]. The complete set of types utilized in this specification along with the notation used to represent these types is listed in Table 12.

Table 12 – Data types

Type	Description
string	For strings listed in this specification, a maximum allowed length can be listed using the form string(N), where N is the maximum string length in characters. For all strings a maximum length is either explicitly indicated or implied by the size of the elements composing the string. For strings in which the content is an enumeration, the longest enumerated value determines the maximum length. If a string does not have an explicitly indicated maximum length or is not an enumeration, the default maximum is 16 characters. Action arguments containing strings longer than the specified maximum MAY result in an “Invalid arguments” error response.
int	Integer in the range –2147483648 to +2147483647, inclusive. For some int types listed, a value range is given using the form int[Min:Max], where the Min and Max values are inclusive. If either Min or Max are missing, this indicates no limit.
unsignedInt	Unsigned integer in the range 0 to 4294967295, inclusive. For some unsignedInt types listed, a value range is given using the form unsignedInt[Min:Max], where the Min and Max values are inclusive. If either Min or Max are missing, this indicates no limit.
boolean	Boolean, where the allowed values are “0”, “1”, “true”, and “false”. The values “1” and “true” are considered interchangeable, where both equivalently represent the logical value *true*. Similarly, the values “0” and “false” are considered interchangeable, where both equivalently represent the logical value *false*.

Type	Description
dateTime	The subset of the ISO 8601 date-time format defined by the SOAP dateTime type. All times MUST be expressed in UTC (Universal Coordinated Time) unless explicitly stated otherwise in the definition of a variable of this type. If absolute time is not available to the CPE, it SHOULD instead indicate the relative time since boot, where the boot time is assumed to be the beginning of the first day of January of year 1, or 0001-01-01T00:00:00. For example, 2 days, 3 hours, 4 minutes and 5 seconds since boot would be expressed as 0001-01-03T03:04:05. Relative time since boot MUST be expressed using an untimezoned representation. Any untimezoned value with a year value less than 1000 MUST be interpreted as a relative time since boot. If the time is unknown or not applicable, the following value representing "Unknown Time" MUST be used: 0001-01-01T00:00:00Z. Any dateTime value other than one expressing relative time since boot (as described above) MUST use UTC timezoned representation (that is, it MUST include a timezone suffix of "Z", "-00:00", or "+00:00") unless explicitly stated otherwise in the definition of the parameter (that is, it MUST include a timezone suffix aligned with the parameter's definition).
base64	Base64 encoded binary. A maximum allowed length can be listed using the form base64(N), where N is the maximum length in characters before Base64 encoding.
anySimpleType	The value of an element defined to be of type "anySimpleType" MAY be of any simple data type, including (but not limited to) any of the other types listed in this table. Following the SOAP specification [12], elements specified as being of type "anySimpleType" MUST include a type attribute to indicate the actual type of the element. For example: <ParameterValueStruct> <Name>Device.DeviceInfo.ProvisioningCode</Name> <Value xsi:type="xsd:string">code12345</Value> </ParameterValueStruct> The namespaces xsi and xsd used above are as defined in [12].

The methods used in this specification also make use of structures and arrays (in some cases containing mixed types). Array elements are indicated with square brackets after the data type. If specified, the maximum length of the array is indicated within the brackets. If the maximum length is not specified, unless otherwise indicated, there is no fixed requirement on the number of elements the recipient will be able to accommodate. A request with an array too large for the recipient to accommodate SHOULD result in the "Resources exceeded" fault code. Unless otherwise specified, the order of items in an array MUST NOT have any effect on the interpretation of the contents of the array.

A.2.2 Instance Identifiers

In some cases, where multiple instances of an Object can occur, the placeholder node name "{i}" is shown. In actual use, this placeholder is to be replaced by an Instance Identifier.

An Instance Identifier is a value that uniquely identifies an instance within a Multi-Instance Object.

An Instance Identifier lifespan is the same as that of its addressed Object instance.

Two types of Instance Identifiers are available: Instance Number and Instance Alias.

A.2.2.1 Instance Number Identifier

The Instance Number identifier allows a Parameter or sub-object within an Object to be referenced by using this Instance Number in the Path Name. The Instance Number assigned by the CPE is arbitrary.

Note – the fact that Instance Numbers are arbitrary means that they do not define a useful Object ordering, e.g. the ACS cannot assume that a newly-created Object will have a higher Instance Number than its existing sibling Objects.

The CPE SHOULD NOT assign an Instance Number that has been used for a previously deleted Object instance. The CPE SHOULD exhaust the full space of integer values for a given Object before re-using Instance Numbers.

Once an Object instance is created, the assigned Instance Numbers MUST persist unchanged until the Object is subsequently deleted (either by the ACS or by a third party). This implies that the Instance Number MUST persist across reboots of the CPE, and that the CPE MUST NOT allow the Instance Number of an existing Object instance to be modified by a third-party entity.

The Instance Number identifier MUST be supported by the CPE.

An Instance Number is expressed as a positive integer (>=1), for example:

```
Device.Services.ABCService.1
```

A.2.2.2 Instance Alias Identifier

This is the Instance Identifier used by the OPTIONAL Alias-Based Addressing Mechanism (see Section 3.6.1). When the Alias-Based Addressing Mechanism is supported, the Path Names within RPC arguments MAY contain Instance Aliases.

An Instance Alias is expressed as a string surrounded in square brackets, for example:

```
Device.Services.ABCService.[a]
```

The square brackets are the notation used to distinguish an Instance Alias from an Instance Number.

The string contained between the square brackets is the value of the addressed Object instance's Alias Parameter (described in Section 3.8/TR-106 [16]). The Instance Alias MUST always be unique within its parent Object and MUST never be empty.

An Instance Alias can be read via its addressed Object instance's Alias Parameter.

An Instance Alias can be changed by modifying its addressed Object instance's Alias Parameter.

A.2.3 Other Requirements

Any message that is sent or received whose arguments do not adhere to the normative CWMP XSD as defined in A.6 MUST generate an error response.

Future versions of this specification MUST NOT alter the RPC method signatures defined in this Annex. Any changes needed in a future version MUST result only in new RPC methods with distinct names being defined.

A.2.4 Instance Wildcards

The use of instance wildcards is OPTIONAL (see 3.6.2). If it is supported the path names within the GPV (see A.3.2.2), GPN (see A.3.2.3) and GPA (see A.3.2.5) RPCs may contain instance wildcards.

An instance wildcard is expressed with the wildcard character "*" instead of an instance identifier in a path name. Path names with multiple instances may contain multiple wildcards. Instance wildcards can be combined with partial paths, with the exception that the wildcard character "*", MUST not be the last part of a path name [21].

Examples for valid path names are:

```
Device.IP.Interface.*.Status
Device.IP.Interface.*. IP4Address.
Device.IP.Interface.*. IP4Address.*.IPAddress
```

These pathnames are invalid:

`Device.IP.Interface.*.`	(Wildcard is last part of path)
`Device.IP.Interface.1.*.IPAddress`	(Wildcard does not replace instance i)

A.3 Baseline RPC Messages

A.3.1 Generic Methods

The methods listed in this Section are REQUIRED to be supported on both CPE devices and ACSs. Either a CPE or ACS MAY call these methods.

A.3.1.1 GetRPCMethods

This method MAY be used by a CPE or ACS to discover the set of methods supported by the ACS or CPE it is in communication with. This list MUST include all the supported methods, both standard methods (those defined in this specification or a subsequent version) and vendor-specific methods. The receiver of the response MUST ignore any unrecognized methods.

Vendor-specific methods MUST be in the form X_<VENDOR>_MethodName, where <VENDOR> is a unique vendor identifier, which MAY be either an OUI or a domain name. The OUI or domain name used for a given vendor-specific method MUST be one that is assigned to the organization that defined this method (which is not necessarily the same as the vendor of the CPE or ACS). An OUI is an organizationally unique identifier as defined in [13], which MUST formatted as a 6 hexadecimal-digit OUI (organizationally unique identifier), with all upper-case letters and any leading zeros included. A domain name MUST be upper case with each dot (".") replaced with a hyphen or underscore. Examples: X_012345_MyMethod, X_ACME_COM_MyMethod.

[21] This is not necessary, because would be redundant to the partial path syntax.

The calling arguments for this method are defined in Table 13. The arguments in the response are defined in Table 14.

Table 13 – GetRPCMethods arguments

Argument	Type	Description
-	void	This method has no calling arguments.

Table 14 – GetRPCMethodsResponse arguments

Argument	Type	Description
`MethodList`	string(64)[]	Array of strings containing the names of each of the RPC methods the recipient supports. The list of methods returned by an ACS MUST always include "Inform". For example, a CPE implementing only the baseline methods defined in this version of the specification would return the following list when requested by an ACS: "GetRPCMethods" "SetParameterValues" "GetParameterValues" "GetParameterNames" "SetParameterAttributes" "GetParameterAttributes" "AddObject" "DeleteObject" "Reboot" "Download" As another example, an ACS implementing only the baseline methods defined in this version of the specification would return the following list when requested by a CPE: "Inform" "GetRPCMethods" "TransferComplete"

The following fault codes are defined for this method for response from a CPE: 9001, 9002.

The following fault codes are defined for this method for response from an ACS: 8001, 8002, 8005.

A.3.2 CPE Methods

The methods listed in this Section are defined to be supported on a CPE device. Only an ACS can call these methods.

A.3.2.1 SetParameterValues

This method MAY be used by an ACS to modify the value of one or more CPE Parameters.

The calling arguments for this method are defined in Table 15. The arguments in the response are defined in Table 16.

Table 15 – SetParameterValues arguments

Argument	Type	Description
`ParameterList`	ParameterValueStruct[]	Array of name-value pairs as specified in Table 17. For each name-value pair, the CPE is instructed to set the Parameter specified by the name to the corresponding value. This array MUST NOT contain more than one entry with the same Parameter name. If a given Parameter appears in this array more than once, the CPE MUST respond with fault 9003 (Invalid arguments). If the length of this array is zero, then the CPE MUST set the ParameterKey to the value specified by the ParameterKey argument, but MUST NOT set any other Parameter values.
`ParameterKey`	string(32)	The value to set the ParameterKey Parameter. The CPE MUST set ParameterKey to the value specified in this argument if and only if SetParameterValues completes successfully. If SetParameterValues does not complete successfully (implying that the Parameter value changes requested did not take effect), the value of ParameterKey MUST NOT be modified. ParameterKey provides the ACS a reliable and extensible means to track changes made by the ACS. The value of this argument is left to the discretion of the ACS, and MAY be left empty.

Table 16 – SetParameterValuesResponse arguments

Argument	Type	Description
`Status`	int[0:1]	A successful response to this method returns an integer enumeration defined as follows: 0 = All Parameter changes have been validated and applied. 1 = All Parameter changes have been validated and committed, but some or all are not yet applied (for example, if a reboot is required before the new values are applied).

If the CPE supports the OPTIONAL Alias-Based Addressing Mechanism (as defined in Section 3.6.1 and described in Appendix II) and its ManagementServer.AutoCreate-Instances Parameter value is set to true, then the Auto-Create Instance Mechanism is performed by the CPE as follows:

- For each Instance Alias identifier supplied in the Path Name that does not already exist, the CPE MUST follow the rules in Section A.3.2.6 for automatically creating the new Object instances.

  Note - The CPE assigned Instance Number is not returned with the SetParameterValuesResponse.

On successful receipt of a SetParameterValues RPC, the CPE MUST apply the changes to all of the specified Parameters atomically. That is, either all of the value changes are applied together, or none of the changes are applied at all. In the latter case, the CPE MUST return a fault response indicating the reason for the failure to apply the changes. The CPE MUST NOT apply any of the specified changes without applying all of them. This requirement MUST hold even if the CPE experiences a crash during the process of applying the changes. The order of Parameters listed in the ParameterList has no significance[22] [23], meaning that the application of value changes to the CPE MUST be independent of the order in which they are listed.

[22] Modification of ManagementServer.AutoCreateInstances or ManagementServer.InstanceMode Parameter(s) will have an undefined effect on Parameters within the same RPC command that are affected by the Auto-Create Instance Mechanism. This is a result of the order of the Parameters processed in the RPC by the CPE having no significance.

[23] For a CPE that supports the Alias-Based Addressing Mechanism, the fact that the order of the Parameters in the

If the request is valid, it is strongly RECOMMENDED that the CPE apply the requested changes prior to sending the SetParameterValues response. If it does so, the CPE MUST set the value of Status in the response to 0 (zero), indicating that the changes have been applied.

If the CPE requires the Session to be terminated before applying some or all of the Parameter values, the CPE MUST reply before all Parameter values have been applied, and thus MUST set the value of Status in the response to 1[24]. In this case, the reply MUST come only after all validation of the request has been completed and the new values have been appropriately saved such that they will definitely be applied as soon as physically possible after the Session has terminated. Once the CPE issues the SetParameterValues response, all changes associated with the corresponding request (including the new ParameterKey) MUST be available for subsequent commands to operate on, regardless of whether the changes have been applied or not. In particular, the use of GetParameterValues to read a Parameter modified by an earlier SetParameterValues MUST return the modified value, even if that value has not yet been applied.

If the value of Status in the SetParameterValues response is 1, the requested changes MUST be applied as soon as physically possible after the Session has terminated, and no later than the beginning of the next Session. Note that if a CPE requires a reboot to cause the changes to be applied, the CPE MUST initiate that reboot on its own after the termination of the Session. Because some CPE will not require a reboot in these circumstances, an ACS SHOULD NOT call the Reboot method as a result of modifying the CPE's configuration, since this would result in an unnecessary reboot. Note also that if application of a configuration change by the CPE would result in a service disruption (for example, if the CPE requires a reboot to apply the requested change), it is not the responsibility of the CPE to avoid or delay such a disruption. To minimize the impact of such a disruption, the ACS MAY delay requesting such a configuration change until an appropriate time, but this is entirely at the ACS's discretion.

The use of the Status value is independent between successive SetParameterValues, AddObject, or DeleteObject requests within the same Session. The use of a Status value of 1 in response to one request does not necessarily imply that subsequent requests in the same Session will also respond in the same way.

The ACS MAY set Parameter values in any combination or order of its choosing using one or multiple SetParameterValues RPCs.

All modifications to a CPE's configuration resulting from use of the SetParameterValues method MUST be retained across reboots of the CPE.

The `ParameterValueStruct` structure is defined in Table 17.

ParameterList has no significance means that the effect is undefined when a SetParameterValues RPC is used to change the value of an Alias Parameter that is also within the same RPC, used in a Parameter name or Parameter value that is a Path Name or a list of Path Names.

[24] When modifying ManagementServer.AutoCreateInstances or ManagementServer.InstanceMode and the CPE returns a committed response (status = 1), all subsequent commands affected by the Alias-Based Addressing Mechanism within the same Session will not reflect the updated mode change(s).

Table 17 – ParameterValueStruct definition

Name	Type	Description
`Name`	string(256)	This is the name of a Parameter. The CPE MUST treat the Parameter name as case sensitive.
`Value`	anySimpleType	This is the value the Parameter is to be set. The CPE MUST treat string-valued Parameter values as case-sensitive.

The following fault codes are defined for this method: 9001, 9002, 9003, 9004, 9005, 9006, 9007 and 9008.

For any Path Name node in the SetParameterValues RPC that is referred to by an Instance Number that does not exist, the CPE MUST return a fault response with Invalid Parameter Name (9005) fault code.

When the OPTIONAL Alias-Based Addressing Mechanism is enabled and its ManagementServer.AutoCreateInstances Parameter value is set to false, then the CPE MUST return a fault response with Invalid Parameter Name (9005) fault code for any Path Name node in the SetParameterValues RPC that is referred to by an Instance Alias that does not exist.

If there is a fault due to one or more Parameters in error, the primary fault code indicated for the overall fault response MUST be Invalid Arguments (9003). The fault response for this method MUST include a SetParameterValuesFault element for each Parameter in error.

The CPE MUST reject an attempt to set values using the SetParameterValues RPC that would result in an invalid configuration, where an invalid configuration is defined as one of the following:

- A Parameter value or combination of Parameter values that are explicitly prohibited in the definition of the Data Model(s) supported by the CPE.
- A Parameter value or combination of Parameter values that are not supported by the CPE.

In both of the above cases, the response from the CPE MUST include a SetParameterValuesFault element for each such Parameter, indicating the Invalid Parameter Value fault code (9007).

The CPE MUST NOT impose any additional configuration restrictions beyond the exceptions described above and restrictions otherwise explicitly permitted or required by the CPE WAN Management Protocol.

A.3.2.2 GetParameterValues

This method MAY be used by an ACS to obtain the value of one or more CPE Parameters. The calling arguments for this method are defined in Table 18. The arguments in the response are defined in Table 19.

Table 18 – GetParameterValues arguments

Argument	Type	Description
`ParameterNames`	string(256)[]	Array of strings, each representing the name of a requested Parameter. If a Parameter name argument is given as a Partial Path Name, the request is to be interpreted as a request to return all of the Parameters in the branch of the naming hierarchy that shares the same prefix as the argument. A Partial Path Name MUST end with a "." (dot) after the last node name in the hierarchy. An empty string indicates the top of the name hierarchy. If the OPTIONAL instance wildcards are used in a parameter name (see 3.6.2) the request is to be interpreted as to return all the parameters of all existing instances, which match the requested parameter name. An instance wildcard may be combined with a partial path name. For the parameter name the rules in A.2.4 apply. Below is an example of a full Parameter name: Device.DeviceInfo.SerialNumber Below is an example of a Partial Path Name: Device.DeviceInfo.

Table 19 – GetParameterValuesResponse arguments

Argument	Type	Description
`ParameterList`	ParameterValueStruct[]	Array of name-value pairs, as specified in Table 17, containing the name and value for each requested Parameter. If multiple entries in the ParameterNames array in the GetParameterValues request overlap such that there are multiple requests for the same Parameter value, it is at the discretion of the CPE whether or not to duplicate that Parameter in the response array. That is, the CPE MAY either include that Parameter value only once in its response, or it MAY include that Parameter value once for each instance that it was requested. If the ParameterNames argument in the request was a Partial Path Name, and if there are no Parameters within the Object represented by that Partial Path Name (at any level below), the ParameterList MUST be empty, and this MUST NOT cause an error response. If the OPTIONAL instance wildcards are used, and there are no matching instances in the object, the ParameterList MUST be empty and this MUST NOT cause an error response.

The following fault codes are defined for this method: 9001, 9002, 9003, 9004, 9005.

If the fault is caused by one or more invalid Parameter names in the ParameterNames array, the Invalid Parameter Name fault code (9005) MUST be used instead of the more general Invalid Arguments fault code (9003). The value of a ParameterNames element MUST be considered invalid if it does not exactly match either the name of a Parameter currently present in the CPE's Data Model (if the ParameterNames element does not end with a dot) or the name of an Object currently present in the CPE's Data Model (if ParameterNames element ends with a dot).

If the fault is caused by using instance wildcards with a CPE not supporting it, the CPE SHOULD return an Invalid Parameter Name fault code (9005).

A.3.2.3 GetParameterNames

This method MAY be used by an ACS to discover the Parameters accessible on a particular CPE. The calling arguments for this method are defined in Table 20. The arguments in the response are defined in Table 21.

Table 20 – GetParameterNames arguments

Argument	Type	Description
`ParameterPath`	string(256)	A string containing either a complete Parameter name, or a Partial Path Name representing a subset of the name hierarchy. An empty string indicates the top of the name hierarchy. A Partial Path Name MUST end with a "." (dot) after the last node name in the hierarchy. If the OPTIONAL instance wildcards are used in a parameter name (see 3.6.2) the request is to be interpreted as to return all the parameters of all existing instances, which match the requested parameter name. An instance wildcard may be combined with a partial path name. For the parameter name the rules in A.2.4 apply. Below is an example of a full Parameter name: Device.DeviceInfo.SerialNumber Below is an example of a Partial Path Name: Device.DeviceInfo.
`NextLevel`	boolean	If false, the response MUST contain the Parameter or Object whose name exactly matches the ParameterPath argument, plus all Parameters and Objects that are descendents of the Object given by the ParameterPath argument, if any (all levels below the specified Object in the Object hierarchy). For example, if ParameterPath were "Device.Hosts.", the response would include the following (if there were a single instance of Host with Instance Number "1"): Device.Hosts. Device.Hosts.HostNumberOfEntries Device.Hosts.Host. Device.Hosts.Host.1. Device.Hosts.Host.1.IPAddress Device.Hosts.Host.1.AddressSource Device.Hosts.Host.1.LeaseTimeRemaining Device.Hosts.Host.1.MACAddress Device.Hosts.Host.1.HostName Device.Hosts.Host.1.InterfaceType Device.Hosts.Host.1.Active If true, the response MUST contain all Parameters and Objects that are next-level children of the Object given by the ParameterPath argument, if any. For example, if ParameterPath were "Device.Hosts.", the response would include the following: Device.Hosts.HostNumberOfEntries Device.Hosts.Host. Or, if ParameterPath were empty, with NextLevel equal true, the response would list only "Device.".

Table 21 – GetParameterNamesResponse arguments

Argument	Type	Description
`ParameterList`	ParameterInfoStruct[]	Array of structures, each containing the name and other information for a Parameter or Object, as defined in Table 22. When NextLevel is false, this list MUST contain the Parameter or Object whose name exactly matches the ParameterPath argument, plus all Parameters and Objects that are descendents of the Object given by the ParameterPath argument, if any (all levels below the specified Object in the Object hierarchy). If the OPTIONAL instance wildcards are used the list MUST contain the parameter or Objects for all instances, whose name matches the ParameterPath argument, plus all the descendents. If the ParameterPath argument is an empty string, names of all Objects and Parameters accessible on the particular CPE are returned. When NextLevel is true, this list MUST contain all Parameters and Object that are next-level children of the Object given by the ParameterPath argument, if any. If the OPTIONAL instance wildcards are used the list MUST contain the parameter or Objects for all instances, whose name matches the ParameterPath argument. For a Parameter, the Name returned in this structure MUST be a full Path Name, ending with the name of the Parameter element. For an Object, the Name returned in this structure MUST be a Partial Path Name, ending with a dot. This list MUST include any Objects that are currently empty. An empty Object is one that contains no instances (for a Multi-Instance Object), no child Objects, and no child Parameters. If NextLevel is true and ParameterPath refers to an Object that is empty, this array MUST contain zero entries. The ParameterList MUST include only Parameters and Objects that are actually implemented by the CPE. If a Parameter is listed, this implies that a GetParameterValues for this Parameter would be expected to succeed.

Table 22 – ParameterInfoStruct definition

Name	Type	Description
`Name`	string(256)	This is the full Path Name of a Parameter or a Partial Path Name.
`Writable`	boolean	Whether or not the Parameter value can be overwritten using the SetParameterValues method. If Name is a Partial Path Name that refers to an Object, this indicates whether or not AddObject can be used to add new instances of this Object. If Name is a Partial Path Name that refers to a particular instance of a Multi-Instance Object, this indicates whether or not DeleteObject can be used to remove this particular instance. This element MUST be true only if the corresponding Parameter or Object as implemented in this CPE is writable as described above. The value of this element MUST reflect only the actual implementation rather than whether or not the specification of the Parameter or Object allows it to be writable.

The following fault codes are defined for this method: 9001, 9002, 9003, 9004, 9005.

If the fault is caused by an invalid ParameterPath value, the Invalid Parameter Name fault code (9005) MUST be used instead of the more general Invalid Arguments fault code (9003). A ParameterPath value MUST be considered invalid if it is not an empty string and does not exactly match a Parameter or Object name currently present in the CPE's Data Model. If NextLevel is true and ParameterPath is a Parameter name rather than a Partial Path Name, the CPE MUST return a fault response with the Invalid Arguments fault code (9003).

If the fault is caused by using instance wildcards in a CPE not supporting it, the CPE SHOULD return an Invalid Parameter Name fault code (9005)

A fault code of 9004 MAY be returned if NextLevel is false and the number of Parameters exceeds the maximum number of Parameters the CPE can handle at once. If NextLevel is true the CPE MUST NOT return the 9004 fault code in any case.

A.3.2.4 SetParameterAttributes

This method MAY be used by an ACS to modify attributes associated with one or more CPE Parameter. The calling arguments for this method are defined in Table 23. The arguments in the response are defined in Table 24.

On successful receipt of a SetParameterAttributes RPC, the CPE MUST apply the changes to all of the specified Parameters immediately and atomically. That is, either all of the attribute changes are applied together, or none of the changes are applied at all. In the latter case, the CPE MUST return a fault response indicating the reason for the failure to apply the changes. The CPE MUST NOT apply any of the specified changes without applying all of them. This requirement MUST hold even if the CPE experiences a crash during the process of applying the changes.

The ACS MAY set Parameter attributes in any combination or order of its choosing using one or multiple SetParameterAttributes RPCs.

If there is more than one entry in the ParameterList array, and the SetParameterAttributes request is successful as described above, the CPE MUST apply the attribute changes in the order of the ParameterList array. That is, if multiple entries in the ParameterList would result in modifying the same attribute of a given Parameter, the attribute value specified later in the ParameterList array MUST overwrite the attribute value specified earlier in the array. This behavior might seem to be inconsistent with that of SetParameterValues, for which it is an error to specify the same Parameter name more than once; this difference is because, unlike SetParameterValues, SetParameterAttributes permits a mixture of full and Partial Paths to be specified.

All modifications to a CPE's configuration resulting from use of the SetParameterAttributes method MUST be retained across reboots of the CPE.

Attributes are associated with actual Parameter instances. If the CPE supports the Alias-Based Addressing Mechanism, when the alias of an instance is changed and an Attribute has been set on a Parameter whose Parameter Path includes that instance, the CPE MUST keep the attribute on the same actual Parameter after the alias change.

When a Parameter is deleted, its attributes MUST also be deleted. Note that this means that if another Parameter with the same Path Name as a previously deleted Parameter is created in the future, this new Parameter will not inherit attributes from the previously deleted Parameter.

A CPE MUST NOT allow any entity other than the ACS to modify attributes of a Parameter.

Table 23 – SetParameterAttributes arguments

Argument	Type	Description
`ParameterList`	SetParameterAttributesStruct[]	List of changes to be made to the attributes for a set of Parameters. Each entry in this array is a SetParameter-AttributesStruct as defined in Table 25. As described above, the order of entries in this array is significant.

Table 24 – SetParameterAttributesResponse arguments

Argument	Type	Description
-	void	This method response has no arguments.

Table 25 – SetParameterAttributesStruct definition

Name	Type	Description
`Name`	string(256)	This is the name of a Parameter to apply the new attributes. Alternatively, this MAY be a Partial Path Name, indicating that the new attributes are to be applied to all Parameters below this point in the naming hierarchy. For such Parameters within Multi-Instance Objects where the Instance Identifier is below the specified point in the naming hierarchy, the specified attribute values MUST only be applied within instances that exist at the time this method is invoked. A Partial Path Name MUST end with a "." (dot) after the last node name in the hierarchy. An empty string indicates the top of the name hierarchy. Below is an example of a full Parameter name: Device.DeviceInfo.SerialNumber Below is an example of a Partial Path Name: Device.DeviceInfo.
`NotificationChange`	boolean	If true, the value of Notification replaces the current notification setting for this Parameter or group of Parameters. If false, no change is made to the notification setting.
`Notification`	int[0:6]	Indicates whether (and how) the CPE will notify the ACS when the specified Parameter(s) change in value. The following values are defined: 0 = Notification off. The CPE need not inform the ACS of a change to the specified Parameter(s). 1 = Passive notification. Whenever the specified Parameter value changes, the CPE MUST include the new value in the ParameterList in the Non-HEARTBEAT Inform message that is sent the next time a Session is established to the ACS. 2 = Active notification. Whenever the specified Parameter value changes, the CPE MUST initiate a Session to the ACS, and include the new value in the ParameterList in the associated Inform message. 3 = Passive lightweight notification. Whenever the specified Parameter value changes, the CPE MUST include the new value in the ParameterList in the next Lightweight Notification message that is sent. 4 = Passive notification with passive lightweight notification. This combines the requirements of the values 1 (Passive notification) and 3 (Passive lightweight notification). The two mechanisms operate independently. 5 = Active lightweight notification. Whenever the specified Parameter value changes, the CPE MUST include the new value in the ParameterList in the associated Lightweight Notification message and send that message. 6 = Passive notification with active lightweight notification. This combines the requirements of the values 1 (Passive notification) and 5 (Active lightweight notification). The two mechanisms

Name	Type	Description
		operate independently. For Parameters defined in the corresponding Data Model as requiring Forced Active Notification, the value of the Notification attribute is irrelevant and an attempt to set it to a value other than 2 will be ignored. Whenever a Parameter change is sent in the Inform message due to a non-zero Notification setting, the Event code "4 VALUE CHANGE" MUST be included in the list of Events. The delivery of this event and associated parameters is subject to the rules specified in Section 3.7.1.5 (Table 8 in particular). Note that if the CPE deletes an Object containing Parameters for which Notification is enabled (active or passive), this MUST NOT be considered a value-change for the purpose of Notification. By default, prior to any changes to this attribute by an ACS, its value SHOULD be 0 (Notification off) unless otherwise specified in the appropriate Data Model definition. The CPE MAY provide no support for Active notification or Active lightweight notification on a Parameter deemed inappropriate for Active notification. A Parameter is deemed inappropriate for Active notification if and only if that Parameter is explicitly defined as such in the definition of the corresponding Data Model. Parameters that might be deemed inappropriate for Active notification include Parameters that change frequently, such as statistics. A CPE MUST accept a request to enable Passive notification for any Parameter. If lightweight notifications are supported (i.e. Notification values 3, 4, 5, and 6), the CPE MUST additionally accept requests to enable Passive lightweight notification for any Parameter. Note that if a CPE implementation does not allow a particular Parameter value to change in a manner that would result in a Notification (e.g., a capability flag that could only change as a result of a firmware update that requires a reboot, or a writeable Parameter that can only be modified via the CPE WAN Management Protocol), then support for Notification for this Parameter involves no more than keeping track of the value of its Notification attribute. For such a Parameter, the CPE implementation need not incorporate a mechanism to detect value changes nor to initiate Notifications based on such changes.
`AccessListChange`	boolean	If true, the value of AccessList replaces the current access list for this Parameter or group of Parameters. If false, no change is made to the access list.

Name	Type	Description
`AccessList`	string(64)[]	Array of zero or more entities for which write access to the specified Parameter(s) is granted. If there are no entries, write access is only allowed from an ACS. At present, only one type of entity is defined that can be included in this list: “Subscriber” Indicates write access by an interface controlled on the subscriber LAN. Includes any and all such LAN-side mechanisms, which MAY include but are not limited to TR-064 (LAN-side DSL CPE Configuration Protocol), UPnP, the device’s user interface, client-side telnet, and client-side SNMP. Currently, access restrictions for other WAN-side configuration protocols is not specified. The ACS MAY further specify management entities in the ACL using a vendor-specific prefix. If such entities are specified by vendors, they MUST be preceded by X_<VENDOR>_and follow the syntax for vendor extensions for Parameter names defined in [16]. The CPE MUST correctly interpret the value “Subscriber” as described above, but MUST ignore any other individual values in this array that it does not understand. By default, prior to any changes to the access list by an ACS, access SHOULD be granted to all entities specified above. The TR-069 ACS always has write access to all writeable Parameters regardless of being on the access list. Other entities have write access only if they appear on the access list. An entity that is restricted from write access to a certain Parameter MUST NOT be allowed to change Parameter values and MUST NOT be allowed to delete Objects within which the Parameter is contained. The TR-069 access control mechanism does not prevent any entity from creating new Object instances. The CPE MUST accept changes to the AccessList for any Parameter even if that Parameter is read-only and its value cannot be modified by any management entity. For such read-only Parameters, the CPE MUST store the modified AccessList value and return it when requested via GetParameterAttributes, but MAY otherwise ignore this value.

The following fault codes are defined for this method: 9001, 9002, 9003, 9004, 9005, 9009.

If the fault is caused by an invalid Parameter name, the Invalid Parameter Name fault code (9005) MUST be used instead of the more general Invalid Arguments fault code (9003).

If the CPE does not support Active notifications on a Parameter deemed inappropriate (as described above), it MUST reject an attempt to enable an Active notification for that Parameter by responding with fault 9009 (Notification request rejected). If Active notification is being enabled for Parameter(s) specified via a Partial Path Name and the CPE does not support Active notification for one or more such Parameters deemed inappropriate below this point in the naming hierarchy, the CPE MUST reject the request and respond with fault code 9009 (Notification request rejected).

If the CPE does not support the lightweight notification mechanism, it MUST reject any attempt to enable Passive or Active lightweight notifications (codes 3, 4, 5, 6) with fault code 9003 (Invalid arguments).

A.3.2.5 GetParameterAttributes

This method MAY be used by an ACS to read the attributes associated with one or more CPE Parameter. The calling arguments for this method are defined in Table 26. The arguments in the response are defined in Table 27.

Table 26 – GetParameterAttributes arguments

Argument	Type	Description
`ParameterNames`	string(256)[]	Array of strings, each representing the name of a requested Parameter. If a Parameter name argument is given as a Partial Path Name, the request is to be interpreted as a request to return all of the Parameters in the branch of the naming hierarchy that shares the same prefix as the argument. A Partial Path Name MUST end with a "." (dot) after the last node name in the hierarchy. An empty string indicates the top of the name hierarchy. If the OPTIONAL instance wildcards are used in a parameter name (see 3.6.2) the request is to be interpreted as to return all the parameters of all existing instances, which match the requested parameter name. An instance wildcard may be combined with a partial path name. For the parameter name the rules in A.2.4 apply. Below is an example of a full Parameter name: Device.DeviceInfo.SerialNumber Below is an example of a Partial Path Name: Device.DeviceInfo.

Table 27 – GetParameterAttributesResponse arguments

Argument	Type	Description
`ParameterList`	ParameterAttributeStruct[]	List of access control information for the specified set of Parameters. Each entry in this array is a ParameterAttributeStruct as defined in Table 25. If multiple entries in the ParameterNames array in the GetParameterAttributes request overlap such that there are multiple requests for the same Parameter attribute, it is at the discretion of the CPE whether or not to duplicate that Parameter in the response array. That is, the CPE MAY either include that Parameter attribute only once in its response, or it MAY include that Parameter attribute once for each instance that it was requested If the ParameterNames argument in the request was a Partial Path Name, and if there are no Parameters within the Object represented by that Partial Path Name (at any level below), the ParameterList MUST be empty, and this MUST NOT cause an error response. If the OPTIONAL instance wildcards are used, and there are no matching instances in the object, the ParameterList MUST be empty and this MUST NOT cause an error response

Table 28 – ParameterAttributeStruct definition

Name	Type	Description
`Name`	string(256)	This is the name of a Parameter to which the attributes are given. The Name MUST be a full Parameter name, and MUST NOT be a Partial Path Name.

Name	Type	Description
`Notification`	int[0:6]	Indicates whether (and how) the CPE will notify the ACS when the specified Parameter(s) change in value. The following values are defined: 0 = Notification off. The CPE need not inform the ACS of a change to the specified Parameter(s). 1 = Passive notification. Whenever the specified Parameter value changes, the CPE MUST include the new value in the ParameterList in the Non-HEARTBEAT Inform message that is sent the next time a Session is established to the ACS. 2 = Active notification. Whenever the specified Parameter value changes, the CPE MUST initiate a Session to the ACS, and include the new value in the ParameterList in the associated Inform message. 3 = Passive lightweight notification. Whenever the specified Parameter value changes, the CPE MUST include the new value in the ParameterList in the next Lightweight Notification message that is sent. 4 = Passive notification with passive lightweight notification. This combines the requirements of the values 1 (Passive notification) and 3 (Passive lightweight notification). The two mechanisms operate independently. 5 = Active lightweight notification. Whenever the specified parameter value changes, the CPE MUST include the new value in the ParameterList in the associated Lightweight Notification message. 6 = Passive notification with active lightweight notification. This combines the requirements of the values 1 (Passive notification) and 5 (Active lightweight notification). The two mechanisms operate independently.
`AccessList`	string(64)[]	Array of zero or more entities for which write access to the specified Parameter(s) is granted. If there are no entries, write access is only allowed from an ACS. At present, only one type of entity is defined that can be included in this list: "Subscriber" Indicates write access by an interface controlled on the subscriber LAN. Includes any and all such LAN-side mechanisms, which MAY include but are not limited to TR-064 (LAN-side DSL CPE Configuration Protocol), UPnP, the device's user interface, client-side telnet, and client-side SNMP. The list MAY include vendor-specific entities, which MUST be preceded by X_<VENDOR>_and follow the syntax for vendor extensions for Parameter names defined in [16]. The ACS MAY ignore any individual items in this array that it does not understand. By default, prior to any changes to the access list by an ACS, the AccessList attribute for all Parameters SHOULD include all entities that the CPE supports, indicating access granted to all of these entities.

The following fault codes are defined for this method: 9001, 9002, 9003, 9004, 9005.

If the fault is caused by an invalid Parameter name, the Invalid Parameter Name fault code (9005) MUST be used instead of the more general Invalid Arguments fault code (9003).

If the fault is caused by using instance wildcards in a CPE not supporting it, the CPE SHOULD return an Invalid Parameter Name fault code (9005)

A.3.2.6 AddObject

This method MAY be used by the ACS to create a new instance of a Multi-Instance Object. The method call takes as an argument the Path Name of the collection of Objects for which a new instance is to be created. For example:

```
Top.Group.Object.
```

This Path Name does not include an Instance Number for the Object to be created. That Instance Number is assigned by the CPE and returned in the response. Once assigned the Instance Number of an Object cannot be changed and persists until the Object is deleted using the DeleteObject method. After creation, Parameters or sub-objects within the Object are referenced by the Path Name appended with the Instance Identifier. For example, if the AddObject method returned an Instance Number of 2, a Parameter within this instance can then be referred to by the path:

```
Top.Group.Object.2.Parameter
```

If the CPE supports the Alias-Based Addressing Mechanism (as defined in Section 3.6.1) then the following are additional requirements:

- The Path Name MAY be followed by an Instance Alias (as defined in Section A.2.2.2) enclosed between square brackets.
- If the Path Name ends with an Instance Alias (enclosed between square brackets) the CPE MUST assign the Instance Alias to the newly created Object instance.
- If the Path Name does not end with an Instance Alias, the CPE MUST assign the newly created Object instance a unique Instance Alias using a ‘cpe-‘ prefix.
- Once assigned, an Instance Alias MUST only be changed by the ACS and it MUST persist until the Object is deleted.

For example, to add an Object instance with its Instance Alias set to “a”:

```
Top.Group.Object.[a].
```

A new Object instance with an Instance Alias “a” will be created. After creation of an Object instance with an Instance Alias, any Parameter within the created Object instance can then be referred to by a Path Name such as:

```
Top.Group.Object.[a].Parameter
```

On creation of an Object using this method, the Parameters contained within the Object MUST be set to their default values and the associated attributes MUST be set to the following:

- Notification is set to zero (notification off) unless otherwise specified in the appropriate Data Model definition.
- AccessList includes all defined entities.

The calling arguments for this method are defined in Table 29. The arguments in the response are defined in Table 30.

Addition of an Object MUST be done atomically. That is, either all of the Parameters and sub-objects are added together, or none are added. In the latter case the CPE MUST return a fault response indicating the reason for the failure to add the Object. The CPE MUST NOT add any contained Parameters or sub-objects as a result of this method call without adding all of them (all Parameters and sub-objects supported by that CPE). This requirement MUST hold even if the CPE experiences a crash during the process of performing the addition.

If the request is valid, it is strongly RECOMMENDED that the CPE apply the Object creation prior to sending the AddObject response. If it does so, the CPE MUST set the value of Status in the response to 0 (zero), indicating that the Object creation has been applied.

If the CPE requires the Session to be terminated before applying the Object creation, the CPE MUST reply before the Object creation has been applied, and thus MUST set the value of Status in the response to 1. In this case, the reply MUST come only after all validation of the request has been completed and the Object creation request has been appropriately saved such that it will definitely be applied as soon as physically possible after the Session has terminated. Once the CPE issues the AddObject response, all changes associated with the corresponding request (including the new ParameterKey) MUST be available for subsequent commands to operate on, regardless of whether the changes have been applied or not. In particular, even if the Object creation has not yet been applied, the CPE MUST allow the use of SetParameterValues, GetParameterValues, SetParameterAttributes, and GetParameterAttributes to operate on Parameters within the newly created Object, as well as the use of AddObject to create a sub-object within the newly created Object, and DeleteObject to delete either a sub-object or the newly created Object itself.

If the value of Status in the AddObject response is 1, the requested Object creation MUST be applied as soon as physically possible after the Session has terminated, and no later than the beginning of the next Session. Note that if a CPE requires a reboot to cause the Object creation to be applied, the CPE MUST initiate that reboot on its own after the termination of the Session. Because some CPE will not require a reboot in these circumstances, an ACS SHOULD NOT call the Reboot method as a result of modifying the CPE's configuration, since this would result in an unnecessary reboot. Note also that if application of a configuration change by the CPE would result in a service disruption (for example, if the CPE requires a reboot to apply the requested change), it is not the responsibility of the CPE to avoid or delay such a disruption. To minimize the impact of such a disruption, the ACS MAY delay requesting such a configuration change until an appropriate time, but this is entirely at the ACS's discretion.

The use of the Status value is independent between successive SetParameterValues, AddObject, or DeleteObject requests within the same Session. The use of a Status value of 1 in response to one request does not necessarily imply that subsequent requests in the same Session will also respond in the same way.

All modifications to a CPE's configuration resulting from use of the AddObject method MUST be retained across reboots of the CPE. This MUST include the values of Object Instance Identifiers.

Table 29 – AddObject arguments

Argument	Type	Description
`ObjectName`	string(256)	The Path Name of the collection of Objects for which a new instance is to be created. The Path Name MUST end with a "." (dot) after the last node in the hierarchical name of the Object, or if CPE supports the Alias-Based Addressing Mechanism, it MAY end with the requested Instance Alias name for the new Object enclosed between square brackets and MUST end with a ".".
`ParameterKey`	string(32)	The value to set the ParameterKey Parameter. The CPE MUST set ParameterKey to the value specified in this argument if and only if AddObject completes successfully. If AddObject does not complete successfully (implying that the requested Object did not get added), the value of ParameterKey MUST NOT be modified. ParameterKey provides the ACS a reliable and extensible means to track changes made by the ACS. The value of this argument is left to the discretion of the ACS, and MAY be left empty.

Table 30 – AddObjectResponse arguments

Argument	Type	Description
`InstanceNumber`	UnsignedInt[1:]	The Instance Number of the newly created Object. Once created, a Parameter or sub-object within this Object can be later referenced by using this Instance Number Identifier (defined in Section A.2.2.1) in the Path Name. The Instance Number assigned by the CPE is arbitrary. Note – the fact that Instance Numbers are arbitrary means that they do not define a useful Object ordering, e.g. the ACS cannot assume that a newly-created Object will have a higher Instance Number than its existing sibling Objects.
`Status`	int[0:1]	A successful response to this method returns an integer enumeration defined as follows: 0 = The Object has been created. 1 = The Object creation has been validated and committed, but not yet applied (for example, if a reboot is required before the new Object can be applied).

The following fault codes are defined for this method: 9001, 9002, 9003, 9004 and 9005.

If an AddObject request would result in exceeding the maximum number of such Objects supported by the CPE, the CPE MUST return a fault response with the Resources Exceeded (9004) fault code. If an AddObject request uses an Instance Alias and requests a new Object instance and the Instance Alias already exists, the CPE MUST return a fault response with "Invalid Parameter Name" (9005) fault code. The same fault code (9005) MUST be returned if the AddObject request uses an Instance Alias and the instance does not have an Alias Parameter.

A.3.2.7 DeleteObject

This method is used to remove a particular instance of an Object. This method call takes as an argument the Path Name of the Object instance including the Instance Identifer. For example:

```
Top.Group.Object.2.
```

If this method call is successful, the specified instance of this Object is subsequently unavailable for access and the CPE MUST discard the state previously associated with all Parameters (values and attributes) and sub-objects contained within this instance.

When an Object instance is deleted, the Instance Numbers associated with any other instances of the same collection of Objects remain unchanged. Thus, the Instance Numbers of Object instances in a collection might not be consecutive.

The calling arguments for this method are defined in Table 31. The arguments in the response are defined in Table 32.

If the request is valid, it is strongly RECOMMENDED that the CPE apply the Object deletion prior to sending the DeleteObject response. If it does so, the CPE MUST set the value of Status in the response to 0 (zero), indicating that the Object deletion has been applied.

If the CPE requires the Session to be terminated before applying the Object deletion, the CPE MUST reply before the Object deletion has been applied, and thus MUST set the value of Status in the response to 1. In this case, the reply MUST come only after all validation of the request has been completed and the Object deletion request has been appropriately saved such that it will definitely be applied as soon as physically possible after the Session has terminated. Once the CPE issues the DeleteObject response, all changes associated with the corresponding request (including the new ParameterKey) MUST be available for subsequent commands to operate on, regardless of whether the changes have been applied or not. In particular, the use of GetParameterNames and GetParameterValues MUST indicate the absence of the deleted Object, and any attempt to modify or read Parameters or sub-objects within the deleted Object MUST fail.

If the value of Status in the DeleteObject response is 1, the requested Object deletion MUST be applied as soon as physically possible after the Session has terminated, and no later than the beginning of the next Session. Note that if a CPE requires a reboot to cause the Object deletion to be applied, the CPE MUST initiate that reboot on its own after the termination of the Session. Because some CPE will not require a reboot in these circumstances, an ACS SHOULD NOT call the Reboot method as a result of modifying the CPE’s configuration, since this would result in an unnecessary reboot. Note also that if application of a configuration change by the CPE would result in a service disruption (for example, if the CPE requires a reboot to apply the requested change), it is not the responsibility of the CPE to avoid or delay such a disruption. To minimize the impact of such a disruption, the ACS MAY delay requesting such a configuration change until an appropriate time, but this is entirely at the ACS’s discretion.

The use of the Status value is independent between successive SetParameterValues, AddObject, or DeleteObject requests within the same Session. The use of a Status value of 1 in response to one request does not necessarily imply that subsequent requests in the same Session will also respond in the same way.

On deletion, all Parameters and sub-objects contained within this Object MUST be removed atomically. That is, either all of the Parameters and sub-objects are removed together, or none are removed at all. In the latter case, the CPE MUST return a fault response indicating the reason for the failure to delete the Object. The CPE MUST NOT remove any contained Parameters or sub-objects as a result of this method call without removing all of them. This requirement MUST hold even if the CPE experiences a crash during the process of performing the deletion.

All modifications to a CPE's configuration resulting from use of the DeleteObject method MUST be retained across reboots of the CPE.

Table 31 – DeleteObject arguments

Argument	Type	Description
`ObjectName`	string(256)	The Path Name of the Object instance to be removed. The Path Name MUST end with a "." (dot) after the Instance Identifier of the Object.
`ParameterKey`	string(32)	The value to set the ParameterKey Parameter. The CPE MUST set ParameterKey to the value specified in this argument if and only if DeleteObject completes successfully. If DeleteObject does not complete successfully (implying that the requested Object did not get deleted), the value of ParameterKey MUST NOT be modified. ParameterKey provides the ACS a reliable and extensible means to track changes made by the ACS. The value of this argument is left to the discretion of the ACS, and MAY be left empty.

Table 32 – DeleteObjectResponse arguments

Argument	Type	Description
`Status`	int[0:1]	A successful response to this method returns an integer enumeration defined as follows: 0 = The Object has been deleted. 1 = The Object deletion has been validated and committed, but not yet applied (for example, if a reboot is required before the Object can be deleted).

The following fault codes are defined for this method: 9001, 9002, 9003, 9005.

If the fault is caused by an invalid ObjectName value, the Invalid Parameter Name fault code (9005) MUST be used instead of the more general Invalid Arguments fault code (9003). The ObjectName value MUST be considered invalid if it does not exactly match the name of a single instance of a Multi-Instance Object currently present in the CPE's Data Model.

A.3.2.8 Download

Note – The functionality provided by this method overlaps that of the ScheduleDownload method [Section A.4.1.8]. Unlike ScheduleDownload, this method does not provide fine-grained control over when the download can be performed and applied. Also, this method permits a file to be downloaded and applied within the same Session.

This method MAY be used by the ACS to cause the CPE to download a specified file from the designated location. The calling arguments for this method are defined in Table 33. The arguments in the response are defined in Table 34.

When a download is initiated using this method, the CPE MUST indicate successful or unsuccessful completion of the download using one of the following three means:

- A DownloadResponse with the Status argument having a value of zero (indicating success), or a fault response to the Download request (indicating failure).
- A TransferComplete message sent later in the same Session as the Download request (indicating either success or failure). In this case, the Status argument in the corresponding DownloadResponse MUST have a value of one.
- A TransferComplete message sent in a subsequent Session (indicating either success or failure). In this case, the Status argument in the corresponding DownloadResponse MUST have a value of one.

Regardless of which means is used, the CPE MUST only indicate successful completion of the download after the downloaded file(s) has been both successfully transferred and applied. While the criterion used to determine when a file has been successfully applied is specific to the CPE's implementation, the CPE SHOULD consider a downloaded file to be successfully applied only after the file is installed and in use as intended.

In the particular case of downloading a software image, as indicated by a "1 Firmware Upgrade Image" FileType, the CPE MUST consider the downloaded file(s) to be successfully applied only after the new software image is actually installed and operational. If the software image replaces the overall software of the CPE (which would typically require a reboot to install and begin execution), the SoftwareVersion represented in the Data Model MUST already reflect the updated software image in the Session in which the CPE sends a TransferComplete indicating successful download.

In the particular case of downloading a software image via the "6 Stored Firmware Image" FileType, the CPE MUST consider the downloaded file(s) to be successfully applied when they are downloaded, validated, and installed only. That is, the new firmware image does not actually have to be running in order for the device to consider the download to be completed.

If the CPE requires a reboot to apply the downloaded file, then the only appropriate means of indicating successful completion is the third option listed above—a TransferComplete message sent in a subsequent Session.

If the file cannot be successfully downloaded or applied, the CPE MUST NOT attempt to retry the file download on its own initiative, but instead MUST report the failure of the download to the ACS using any of the three means listed above. Upon the ACS being informed of the failure of a download, the ACS MAY subsequently attempt to reinitiate the download by issuing a new Download request.

If the CPE receives one or more Download or ScheduleDownload requests before performing a previously requested download, the CPE MUST queue all requested downloads and perform each of them as closely as possible to the requested time (based on the value of the DelaySeconds argument and the time of the request). Queued downloads MUST be retained across reboots of the CPE. The CPE MUST be able to queue a minimum of three file transfers (downloads and uploads).

For each download performed, the CPE MUST send a distinct TransferComplete. Note that the order in which a series of requested downloads will be performed might differ from the order of the corresponding requests due to differing values of DelaySeconds. For example, an ACS could request a download with DelaySeconds equal to one hour, then five minutes later request a second download with DelaySeconds equal to one minute. In this case, the CPE would perform the second download before the first.

All modifications to a CPE's configuration resulting from use of the Download method MUST be retained across reboots of the CPE.

Table 33 – Download arguments

Argument	Type	Description
`CommandKey`	string(32)	The string the CPE uses to refer to a particular download. This argument is referenced in the methods Inform, TransferComplete, GetQueuedTransfers, GetAllQueuedTransfers and CancelTransfer. The value of the CommandKey is entirely at the discretion of the ACS and MAY be an empty string.
`FileType`	string(64)	An integer followed by a space followed by the file type description. Only the following values are currently defined for the FileType argument: "1 Firmware Upgrade Image" "2 Web Content" “3 Vendor Configuration File” “4 Tone File” (see [28] Appendix B) “5 Ringer File” (see [28] Appendix B) “6 Stored Firmware Image” (see Appendix V) The following format is defined to allow the unique definition of vendor-specific file types: "X <VENDOR> <Vendor-specific identifier>" <VENDOR> is replaced by a unique vendor identifier, which MAY be either an OUI or a domain name. The OUI or domain name used for a given vendor-specific file type MUST be one that is assigned to the organization that defined this method (which is not necessarily the same as the vendor of the CPE or ACS). An OUI is an organizationally unique identifier as defined in [13], which MUST be formatted as a 6 hexadecimal-digit OUI (organizationally unique identifier), with all upper-case letters and any leading zeros included. A domain name MUST be upper case with each dot (“.”) replaced with a hyphen or underscore. If and only if the CPE supports downloading of firmware images using the Download method, the CPE MUST support the "1 Firmware Upgrade Image" FileType value. The definition of a firmware upgrade image MAY vary across different CPE vendors, ranging from a single monolithic image to a set of inter-dependent files, but MUST be presented as a single URL to the CPE. For example, a URL could be a file that contains multiple URLs and instructions on how to upgrade the firmware image. All other FileType values are OPTIONAL. The FileType value of "2 Web Content" is intended to be used for downloading files that contain only web content for a CPE’s web-based user interface. A CPE that supports a web-based user interface and allows the content to be downloaded from the ACS via the Download method as a distinct file containing only web content SHOULD use the FileType value of "2 Web Content" when performing such a download. A CPE that supports a web-based user interface and allows the content to be downloaded from the ACS MAY instead include web content as part of its firmware upgrade image, or use some other means to update the web content in the CPE. Such a CPE need not support the FileType value of "2 Web Content". The FileType value of “3 Vendor Configuration File” is intended to be used for downloading a single vendor configuration file. A CPE MAY instead include one or more vendor configuration files as part of its firmware upgrade image.
`URL`	string(256)	URL, as defined in [15], specifying the source file location. HTTP and HTTPS transports MUST be supported. Other optional transports, as specified in Section 2.3.2, MAY be supported. If the CPE receives multiple Download requests with the same source URL, the CPE MUST perform each download as requested, and MUST NOT assume that the content of the file to be downloaded is the same each time. This URL MUST NOT include the “userinfo” component, as defined in [15].
`Username`	string(256)	Username to be used by the CPE to authenticate with the file server. This string is set to the empty string if no authentication is required.
`Password`	string(256)	Password to be used by the CPE to authenticate with the file server. This string is set to the empty string if no authentication is required.

Argument	Type	Description
`FileSize`	unsignedInt	The size of the file to be downloaded in bytes. The FileSize argument is intended as a hint to the CPE, which the CPE MAY use to determine if it has sufficient space for the file to be downloaded, or to prepare space to accept the file. The ACS MAY set this value to zero. The CPE MUST interpret a zero value to mean that that the ACS has provided no information about the file size. In this case, the CPE MUST attempt to proceed with the download under the presumption that sufficient space is available, though during the course of download, the CPE might determine otherwise. The ACS SHOULD set the value of this Parameter to the exact size of the file to be downloaded. If the value is non-zero, the CPE MAY reject the Download request on the basis of insufficient space. If the CPE attempts to proceed with the download based on the value of this argument, but the actual file size differs from the value of this argument, this could result in a failure of the download. However, the CPE MUST NOT cause the download to fail solely because it determines that the value of this argument is inaccurate.
`TargetFileName`	string(256)	The name of the file to be used on the target file system. This argument MAY be left empty if the target file name can be extracted from the downloaded file itself, or from the URL argument, or if no target file name is needed. If this argument is specified, but the target file name is also indicated by another source (for example, if it is extracted from the downloaded file itself), this argument MUST be ignored. If the target file name is used, the downloaded file would replace any existing file of the same name (whether or not the CPE archives the replaced file is a local matter). If present, this Parameter is treated as an opaque string with no specific requirements for its format. That is, the TargetFileName value is to be interpreted based on the CPE's vendor-specific file naming conventions. Note that this specification does not preclude the use of a file naming convention in which the file's path can be specified as part of the file name.

Argument	Type	Description
`DelaySeconds`	unsignedInt	This argument has different meanings for Unicast and Multicast downloads. For Unicast downloads it is the number of seconds before the CPE will initiate the download. For Multicast downloads the CPE will initiate the download immediately and it is the number of seconds available for initiating, performing and applying the download. The following applies only to Unicast downloads, i.e. to downloads where the URL specifies a Unicast download transport protocol. The number of seconds from the time this method is called to the time the CPE is requested to initiate the download. A value of zero indicates that no delay is requested. If a non-zero delay is requested, the download MUST NOT occur in the same Session in which the request was issued. The CPE MUST perform and apply the download immediately after the time indicated by DelaySeconds, unless this is not possible for reasons outside the CPE's control, in which case the CPE MUST attempt to perform and apply the download within one hour after the time indicated by DelaySeconds. If the CPE cannot begin the download within this time window, the CPE MUST consider the download to have failed and report this failure to the ACS using the TransferComplete method. If the download completes before the end of this time window, the CPE MUST apply the download prior to the end of this time window. If the download is still in progress at the end of this time window, the CPE MUST apply the download immediately upon completion of the download. The following applies only to Multicast downloads, i.e. to downloads where the URL specifies a Multicast download transport protocol: The number of seconds from the time this method is called that are available for the CPE to initiate, perform and apply the download. Multicast downloads MUST NOT occur in the same Session in which the request was issued. The CPE MUST perform and apply the download immediately, unless this is not possible for reasons outside the CPE's control, in which case the CPE MUST attempt to perform and apply the download within DelaySeconds of the download request. If the CPE cannot complete the download within this time window, the CPE MUST consider the download to have failed and report this failure to the ACS using the TransferComplete method. The following applies to both Unicast and Multicast downloads: The CPE MUST attempt to perform the download within the time window specified above even if the CPE reboots one or more times prior to that time.
`SuccessURL`	string(256)	When applicable, this argument contains the URL, as defined in [15], the CPE SHOULD redirect the user's browser to if the download completes successfully. This URL MAY include CGI arguments (for example, to maintain session state). This applies only if the download was initiated via browser-based user interaction and the CPE supports the ability to selectively redirect based on the download results. When there is no need for such a URL, this argument SHOULD be empty.
`FailureURL`	string(256)	When applicable, this argument contains the URL, as defined in [15], the CPE SHOULD redirect the user's browser to if the download does not complete successfully. This URL MAY include CGI arguments (for example, to maintain session state). This applies only if the download was initiated via browser-based user interaction and the CPE supports the ability to selectively redirect based on the download results. When there is no need for such a URL, this argument SHOULD be empty.

Table 34 – DownloadResponse arguments

Argument	Type	Description
`Status`	int[0:1]	A successful response to this method returns an integer enumeration defined as follows: 0 = Download has completed and been applied. 1 = Download has not yet been completed and applied (for example, if the CPE needs to reboot itself before it can perform the file download, or if the CPE needs to reboot itself before it can apply the downloaded file). If the value of this argument is non-zero, the CPE MUST subsequently call the TransferComplete method to indicate the completion status of this download (either successful or unsuccessful) either later in the same Session or in a subsequent Session.
`StartTime`	dateTime	The date and time download was started in UTC. This need only be filled in if the download has been completed. Otherwise, the value MUST be set to the Unknown Time value.
`CompleteTime`	dateTime	The date and time the download was fully completed and applied in UTC. This need only be filled in if the download has been completed. Otherwise, the value MUST be set to the Unknown Time value.

The following fault codes are defined for this method: 9000, 9001, 9002, 9003, 9004, 9010, 9012, 9013.

If an attempt is made to queue an additional download when the CPE's file transfer queue is already full, the CPE MUST respond with fault 9004 (Resources exceeded). If the CPE detects the presence of the "userinfo" component in the file source URL, it SHOULD reject the Download request with the fault code 9003 (Invalid arguments). If the CPE rejects the Download request because the FileSize argument exceeds the available space on the device, it MUST use the Download Failure (9010) fault code.

A.3.2.9 Reboot

This method causes the CPE to reboot, and calls for use of extreme caution. The CPE MUST send the method response and complete the remainder of the Session prior to rebooting. The calling arguments for this method are defined in Table 35. The arguments in the response are defined in Table 36.

Note – Multiple invocations of this method within a single Session MUST result in only a single reboot. In this case the Inform following the reboot would be expected to contain a single "1 BOOT" EventCode and an "M Reboot" EventCode for each method invocation.

This method is primarily intended for troubleshooting purposes. This method is *not* intended for use by an ACS to initiate a reboot after modifying the CPE's configuration (e.g., setting CPE Parameters or initiating a download). If a CPE requires a reboot after its configuration is modified, the CPE MUST initiate that reboot on its own after the termination of the Session[25]. Because some CPE will not require a reboot in these circumstances, an ACS SHOULD NOT call the Reboot method as a result of modifying the CPE's configuration, since this would result in an unnecessary reboot.

[25] The CPE SHOULD wait until all active CWMP Endpoint Sessions are terminated prior to performing the Reboot.

Table 35 – Reboot arguments

Argument	Type	Description
`CommandKey`	string(32)	The string to return in the CommandKey element of the InformStruct when the CPE reboots and calls the Inform method. The value of the CommandKey is entirely at the discretion of the ACS and MAY be an empty string.

Table 36 – RebootResponse arguments

Argument	Type	Description
-	void	This method response has no arguments.

The following fault codes are defined for this method: 9001, 9002, 9003.

A.3.3 ACS Methods

The methods listed in this Section are defined to be supported on an ACS. Only a CPE can call these methods.

A.3.3.1 Inform

A CPE MUST call the `Inform` method to initiate a transaction sequence whenever a Session with an ACS is established. The calling arguments for this method are defined in Table 37. The arguments in the response are defined in Table 38.

Table 37 – Inform arguments

Argument	Type	Value
`DeviceId`	DeviceIdStruct	A structure that uniquely identifies the CPE, defined in Table 39.
`Event`	EventStruct[64]	An array of structures, as defined in Table 8 in Section 3.7.1.5, indicating the events that caused the Session to be established. If one or more causes exist, the CPE MUST list all such causes. The ACS MUST NOT place any significance on the order of events within this array. If a CPE needs to deliver more than 64 events in a single Inform (this would be expected to occur only under exceptional circumstances and on rare occasions), it MUST discard the oldest "M" (method-related) events in order to avoid exceeding the maximum array size. If the Session was established solely because the previous Session terminated unsuccessfully, this array MUST NOT contain events that have already been delivered (if all events have already been delivered this array MUST be empty). If further events occur while a previous failed Session is being retried, the new events MUST be incorporated into the retried Session's event array. If the CPE establishes a Session for which none of the standard event codes apply, then this array MAY be empty.
`MaxEnvelopes`	unsignedInt	This argument MUST be set to a value of 1 because this version of the protocol supports only a single envelope per message, and on reception its value MUST be ignored.
`CurrentTime`	dateTime	The current date and time known to the CPE. This MUST be represented in the local time zone of the CPE, and MUST include the local time-zone offset from UTC (with appropriate adjustment for daylight savings time). How the local time zone is determined by the CPE is beyond the scope of this specification.

Argument	Type	Value
`RetryCount`	unsignedInt	Number of prior times an attempt was made to retry this Session. This MUST be zero if and only if the previous Session, if any, completed successfully, i.e. it will be reset to zero only when a Session completes successfully.
`ParameterList`	ParameterValueStruct[]	Array of name-value pairs as specified in Table 17. This Parameter MUST contain the name-value for the following Parameters: • Every Parameter for which the ACS has set the Notification attribute to either Active notification or Passive notification whose value has been modified by an entity other than the ACS since the last successful Non-HEARTBEAT Inform notification (including values modified by the CPE itself). • Every Parameter defined in the corresponding Data Model as requiring Forced Active Notification (regardless of the value of the Notification attribute) for which the value has been modified by an entity other than the ACS since the last successful Non-HEARTBEAT Inform notification (including values modified by the CPE itself). • Every Parameter defined in the corresponding Data Model as being required in every Inform. If a Parameter has changed more than once since the last successful Inform notification, the Parameter MUST be listed only once, with only the most recent value given. In this case, the Parameter MUST be included in the ParameterList even if its value has changed back to the value it had at the time of the last successful Inform. Whenever the CPE is re-booted, or if the ACS URL is modified, the CPE MAY at that time clear its record of Parameters pending notification due to a value change (though, the CPE MUST retain the values of the Notification attribute for all Parameters). If the CPE clears its record of Parameters pending notification due to a value change, it MUST at the same time discard the corresponding “4 VALUE CHANGE” event. If the value of at least one Parameter listed in the ParameterList has been modified by an entity other than the ACS since the last successful Non-HEARTBEAT Inform notification to the same ACS, the Inform message MUST include the EventCode “4 VALUE CHANGE”. This includes value changes to any of the Parameters that are listed due to being required in every Inform. Otherwise, the Inform message MUST NOT include the EventCode “4 VALUE CHANGE”. If the Inform message does include the “4 VALUE CHANGE” EventCode then the ParameterList MUST include only those Parameters that meet one of the three criteria listed above. If the Inform message does not include the “4 VALUE CHANGE” EventCode, the ParameterList MAY include additional Parameters at the discretion of the CPE. Note that if the Inform message includes the “8 DIAGNOSTICS COMPLETE" EventCode, the CPE is not required to include in the ParameterList any Parameters associated with results of the corresponding diagnostic, and as described above, if the “4 VALUE CHANGE” EventCode is also present in the Inform, the ParameterList MUST include only those Parameters that meet one of the three criteria listed above.

Table 38 – InformResponse arguments

Argument	Type	Description
`MaxEnvelopes`	unsignedInt	This argument MUST be set to a value of 1 because this version of the protocol supports only a single envelope per message, and on reception its value MUST be ignored.

Table 39 – DeviceIdStruct definition

Name	Type	Description
`Manufacturer`	string(64)	Manufacturer of the device (for display only). The value MUST be the same as the value of the DeviceInfo,Manufacturer Parameter.
`OUI`	string(6)	Organizationally unique identifier of the device manufacturer. Represented as a six hexadecimal-digit value using all upper-case letters and including any leading zeros. The value MUST be a valid OUI as defined in [13]. The value MUST be the same as the value of the DeviceInfo.ManufacturerOUI Parameter. This value MUST remain fixed over the lifetime of the device, including across firmware updates. Any change would indicate that it is a new device and would therefore require a BOOTSTRAP Inform.
`ProductClass`	string(64)	Identifier of the class of product for which the serial number applies. That is, for a given manufacturer, this Parameter is used to identify the product or class of product over which the SerialNumber Parameter is unique. The value MUST be the same as the value of the DeviceInfo.ProductClass Parameter. This value MUST remain fixed over the lifetime of the device, including across firmware updates. Any change would indicate that it is a new device and would therefore require a BOOTSTRAP Inform.
`SerialNumber`	string(64)	Identifier of the particular device that is unique for the indicated class of product and manufacturer. The value MUST be the same as the value of the DeviceInfo.SerialNumber Parameter. This value MUST remain fixed over the lifetime of the device, including across firmware updates. Any change would indicate that it is a new device and would therefore require a BOOTSTRAP Inform.

Table 40 – EventStruct definition

Name	Type	Description
`EventCode`	string(64)	Each value consists of an identifying character followed by a text description of the cause. See Table 8 in Section 3.7.1.5 for event codes, handling rules, and a syntax for specifying vendor-specific events. The value of this Parameter is case sensitive and MUST exactly match either one of the values defined in Table 8 in Section 3.7.1.5, or the vendor-specific form also specified in that table.
`CommandKey`	string(32)	If the EventCode in this Event list entry corresponds to a cause in which a CommandKey has been specified, this element MUST contain the value of that CommandKey. For this version of the specification, the following causes result in this argument being set to the value of the CommandKey argument in the originating method call: • ScheduleInform method (EventCode = “M ScheduleInform”) • Reboot method (EventCode = “M Reboot”) • Download method (EventCode = “M Download”) • ScheduleDownload method (EventCode = “M ScheduleDownload”) • ChangeDUState method (EventCode = “M ChangeDUState”) • Upload method (EventCode = “M Upload”) For each of the above methods, the CommandKey value from the method argument MUST appear in the Event array entry containing the EventCode value shown above. For all other EventCode values defined in this specification, the value of CommandKey MUST be an empty string.

The following fault codes are defined for this method: 8001, 8002, 8003, 8004, 8005, 8006.

An ACS that receives an Inform without a "0 BOOTSTRAP" EventCode from a CPE from which it has not previously received an Inform with the "0 BOOTSTRAP" EventCode MAY, at its discretion, respond with a fault code of 8003 (Invalid arguments).

A.3.3.2 TransferComplete

Note – the HoldRequests SOAP Header element is DEPRECATED (see Section 3.4.7), so the ACS SHOULD NOT send it.

This method informs the ACS of the completion (either successful or unsuccessful) of a file transfer initiated by an earlier Download, ScheduleDownload or Upload method call. It MUST NOT be called for a file transfer that has been successfully canceled via a CancelTransfer method call.

This paragraph applies only when the file transfer was initiated via Download or Upload. It does not apply to ScheduleDownload, which does not support downloading within the same Session. TransferComplete MUST be called only when the associated Download or Upload response indicated that the transfer had not yet completed at that time (indicated by a non-zero value of the Status argument in the response). In such cases, it MAY be called either later in the same Session in which the transfer was initiated or in any subsequent Session. Note that in order for it to be called within the same Session in which the transfer was initiated, the CPE will have been sent the InformResponse and Download or Upload request while HoldRequests was true. When used, this method MUST be called only after the transfer has successfully completed, and in the case of a download, the downloaded file has been successfully applied, or after the transfer or apply has failed. If this method fails, the CPE MUST NOT regard the ACS as having been informed of the completion of the file transfer, and MUST attempt to call the method again, either in the current Session or in a new Session, subject to the event delivery rules of Section 3.7.1.5. The calling arguments for this method are defined in Table 41. The arguments in the response are defined in Table 42.

Table 41 – TransferComplete arguments

Argument	Type	Value
`CommandKey`	string(32)	Set to the value of the CommandKey argument passed to CPE in the Download, ScheduleDownload or Upload method call that initiated the transfer.
`FaultStruct`	FaultStruct	A FaultStruct as defined in Table 43. If the transfer was successful, the FaultCode is set to zero. Otherwise a non-zero FaultCode is specified along with a FaultString indicating the failure reason.
`StartTime`	dateTime	The date and time transfer was started in UTC. The CPE SHOULD record this information and report it in this argument, but if this information is not available, the value of this argument MUST be set to the Unknown Time value.
`CompleteTime`	dateTime	The date and time the transfer was fully completed and applied in UTC. This need only be filled in if the transfer has been fully completed and applied. The CPE SHOULD record this information and report it in this argument, but if this information is not available or the transfer has not completed, the value of this argument MUST be set to the Unknown Time value.

Table 42 – TransferCompleteResponse arguments

Argument	Type	Value
-	void	This method response has no arguments.

Table 43 – FaultStruct definition

Name	Type	Value
`FaultCode`	unsignedInt	The numerical fault code as defined in Section A.5.1. In the case of a fault, allowed values are: 9001, 9002, 9010, 9011, 9012, 9014, 9015, 9016, 9017, 9018, 9019, 9020. A value of 0 (zero) indicates no fault.
`FaultString`	string(256)	A human-readable text description of the fault. This field SHOULD be empty if the FaultCode equals 0 (zero).

The following fault codes are defined for this method: 8000, 8001, 8002, 8003, 8004, 8005.

A.3.3.3 AutonomousTransferComplete

This method informs the ACS of the completion (either successful or unsuccessful) of a file transfer that was not specifically requested by the ACS. When used, this method MUST be called only after the transfer has successfully completed, and in the case of a download, the downloaded file has been successfully applied, or after the transfer or apply has failed (e.g. a timeout expired). If this method fails, the CPE MUST NOT regard the ACS as having been informed of the completion of the file transfer, and MUST attempt to call the method again, either in the current Session or in a new Session, subject to the event delivery rules of Section 3.7.1.5. The calling arguments for this method are defined in Table 44. The arguments in the response are defined in Table 45.

Table 44 – AutonomousTransferComplete arguments

Argument	Type	Value
`AnnounceURL`	string(1024)	The URL on which the CPE listened to the announcements that led to this transfer being performed, or an empty string if this transfer was not performed as a result of an announcement, or if no such URL is available.
`TransferURL`	string(1024)	The URL from or to which this transfer was performed, or an empty string if no such URL is available.
`IsDownload`	boolean	Indicates whether the autonomous transfer was a download (true) or an upload (false).

Argument	Type	Value
`FileType`	string(64)	An integer followed by a space followed by the file type description. Only the following values are currently defined for the FileType argument: "1 Firmware Upgrade Image" (download only) "2 Web Content" (download only) "3 Vendor Configuration File" (download or upload) [DEPRECATED for upload] "4 Vendor Log File" (upload only) [DEPRECATED] "4 Tone File" (download only; see [28] Appendix B) "5 Ringer File" (download only; see [28] Appendix B) "6 Vendor Configuration File <i>" (upload only) "7 Vendor Log File <i>" (upload only) "8 Stored Firmware Image" (download only) For "6 Vendor Configuration File <i>", <i> is replaced by the Instance Number from the Vendor Config File Object as defined in the appropriate Root Data Model. The Instance Number corresponds to that of the entry in the vendor config file table that the CPE uploaded. For "7 Vendor Log File <i>", <i> is replaced by the Instance Number from the Vendor Log File Object as defined in the appropriate Root Data Model. The Instance Number corresponds to that of the entry in the vendor log file table that the CPE uploaded. The following format is defined to allow the unique definition of vendor-specific file types: "X <VENDOR> <Vendor-specific identifier>" <VENDOR> is replaced by a unique vendor identifier, which MAY be either an OUI or a domain name. The OUI or domain name used for a given vendor-specific file type MUST be one that is assigned to the organization that defined this method (which is not necessarily the same as the vendor of the CPE or ACS). An OUI is an organizationally unique identifier as defined in [13], which MUST be formatted as a 6 hexadecimal-digit OUI (organizationally unique identifier), with all upper-case letters and any leading zeros included. A domain name MUST be upper case with each dot (".") replaced with a hyphen or underscore.
`FileSize`	unsignedInt	The size of the file in bytes, or zero if this information is not available or if the CPE chooses not to make it available.
`TargetFileName`	string(256)	The name of the file on the target (CPE) file system, or an empty string if this information is not available or if the CPE chooses not to make it available.
`FaultStruct`	FaultStruct	A FaultStruct as defined in Table 40. If the transfer was successful, the FaultCode is set to zero. Otherwise a non-zero FaultCode is specified along with a FaultString indicating the failure reason.
`StartTime`	dateTime	The date and time transfer was started in UTC. The CPE SHOULD record this information and report it in this argument, but if this information is not available, the value of this argument MUST be set to the Unknown Time value.
`CompleteTime`	dateTime	The date and time the transfer was fully completed and applied in UTC. The CPE SHOULD record this information and report it in this argument, but if this information is not available, the value of this argument MUST be set to the Unknown Time value.

Table 45 – AutonomousTransferCompleteResponse arguments

Argument	Type	Value
-	void	This method response has no arguments.

The following fault codes are defined for this method: 8000, 8001, 8002, 8003, 8004, 8005.

A.4 Optional RPC Messages

A.4.1 CPE Methods

The methods listed in this Section MAY optionally be supported on a CPE device. Only an ACS can call these methods.

A.4.1.1 GetQueuedTransfers

Note – this method is DEPRECATED in favor of GetAllQueuedTransfers [Section A.4.1.7].

This method MAY be used by an ACS to determine the status of previously requested downloads or uploads. The calling arguments for this method are defined in Table 46. The arguments in the response are defined in Table 47.

Table 46 – GetQueuedTransfers arguments

Argument	Type	Description
-	void	This method has no calling arguments.

Table 47 – GetQueuedTransfersResponse arguments

Argument	Type	Description
`TransferList`	QueuedTransferStruct[16]	Array of structures as defined in Table 48, each describing the state of one transfer that the CPE has been instructed to perform, but has not yet been fully completed.

Table 48 – QueuedTransferStruct definition

Name	Type	Description
`CommandKey`	string(32)	Set to the value of the CommandKey argument passed to CPE in the Download or Upload method call that initiated the transfer.
`State`	int[1:3]	The current state of the transfer. Defined values are: 1 = Not yet started 2 = In progress 3 = Completed, finishing cleanup All other values are reserved.

The following fault codes are defined for this method: 9000, 9001, 9002.

A.4.1.2 ScheduleInform

This method MAY be used by an ACS to request the CPE to schedule a one-time Inform method call (separate from its periodic Inform method calls) sometime in the future. The calling arguments for this method are defined in Table 49. The arguments in the response are defined in Table 50.

If the CPE receives one or more ScheduleInform requests before performing a previously requested ScheduleInform, the CPE MUST queue all requested ScheduleInform requests and perform each of them as closely as possible to the requested time (based on the value of the DelaySeconds argument and the time of the request). Queued ScheduleInform requests MUST be retained across reboots of the CPE if and only if the CPE supports absolute time. The CPE MUST be able to queue a minimum of three ScheduleInform requests. The CPE MUST consider a ScheduleInform that has the same non-empty

CommandKey as a previously requested (and still queued) ScheduleInform as an update to the DelaySeconds of the previously requested ScheduleInform.

Table 49 – ScheduleInform arguments

Argument	Type	Description
`DelaySeconds`	unsignedInt	The number of seconds from the time this method is called to the time the CPE is requested to initiate a one-time Inform method call. The CPE sends a response, and then DelaySeconds later calls the Inform method. This argument MUST be greater than zero.
`CommandKey`	string(32)	The string to return in the CommandKey element of the InformStruct when the CPE calls the Inform method. The value of the CommandKey is entirely at the discretion of the ACS and MAY be an empty string.

Table 50 – ScheduleInformResponse arguments

Argument	Type	Description
-	void	This method response has no arguments.

The following fault codes are defined for this method: 9000, 9001, 9002, 9003.

A.4.1.3 SetVouchers

Note – this method, as part of the "voucher mechanism" as defined in Annex C, is DEPRECATED in favor of the "Software Module Management mechanism" as described in Appendix II / TR-157 Amendment 3 [32].

This method MAY be used by an ACS to set one or more option Vouchers in the CPE. The calling arguments for this method are defined in Table 51. The arguments in the response are defined in Table 52.

Table 51 – SetVouchers arguments

Argument	Type	Description
`VoucherList`	base64[]	Array of Vouchers, where each Voucher is represented as a Base64 encoded octet string. The detailed structure of a Voucher is defined in Annex C.

Table 52 – SetVouchersResponse arguments

Argument	Type	Description
-	void	This method response has no arguments.

The following fault codes are defined for this method: 9000, 9001, 9002, 9003, 9004.

A.4.1.4 GetOptions

Note – this method, as part of the "voucher mechanism" as defined in Annex C, is DEPRECATED in favor of the "Software Module Management mechanism" as described in Appendix II / TR-157 Amendment 3 [32].

This method MAY be used by an ACS to obtain a list of the options currently set in a CPE, and their associated state information. The calling arguments for this method are defined in Table 53. The arguments in the response are defined in Table 54.

Table 53 – GetOptions arguments

Argument	Type	Description
`OptionName`	string(64)	A string representing either the name of a particular Option, or an empty string indicating the method SHOULD return the state of all Options supported by the CPE (whether or not they are currently enabled).

Table 54 – GetOptionsResponse arguments

Argument	Type	Description
`OptionList`	OptionStruct[]	Array of OptionStructs as defined in Table 55, containing either a single OptionStruct if information about a particular Option was requested, or a list of OptionStructs, one for each option supported by the CPE.

Table 55 – OptionStruct definition

Name	Type	Description
`OptionName`	string(64)	Identifying name of the particular Option.
`VoucherSN`	unsignedInt	Identifying number of the particular Option.
`State`	unsignedInt	A number formed by two bits, defined as follows: Bit 0 (LSB): 0 = Option is currently disabled 1 = Option is currently enabled Bit 1: 0 = Option has not been setup 1 = Option has been setup The interpretation of the setup state of an Option is Option-specific, but in general is to be interpreted as indicating whether the end-user has actively performed any actions required to make the Option fully operational.
`Mode`	int[0:2]	This element specifies whether the designated Option is enabled or disabled; and if enabled, whether or not an expiration has been specified. The defined values are: 0 = Disabled 1 = Enabled with expiration 2 = Enabled without expiration
`StartDate`	dateTime	The specified start date for the Option in UTC. If in the future, this is the date the Option is to be enabled. If in the past, this is the date the Option was enabled. This element applies only when the value of the Mode element is 1 (Enabled with expiration). When the Mode element has any other value, StartDate MUST be set to the Unknown Time value.
`ExpirationDate`	dateTime	The specified date the Option is to expire in UTC, if any. This element applies only when the value of the Mode element is 1 (Enabled with expiration). When the Mode element has any other value, ExpirationDate MUST be set to the Unknown Time value.
`IsTransferable`	boolean	Indicates whether or not the Option has been designated transferable or non-transferable (see Annex C). Defined values are: 0 = Non-transferable 1 = Transferable

The following fault codes are defined for this method: 9000, 9001, 9002, 9003.

A.4.1.5 Upload

This method MAY be used by the ACS to cause the CPE to upload a specified file to the designated location. The calling arguments for this method are defined in Table 56. The arguments in the response are defined in Table 57.

If the file cannot be successfully uploaded, the CPE MUST NOT attempt to retry the file upload on its own initiative, but instead MUST report the failure of the upload to the ACS via either the Upload response (if it has not yet been sent) or the TransferComplete method. Upon the ACS being informed of the failure of an upload, the ACS MAY subsequently attempt to reinitiate the upload by issuing a new Upload request.

If the CPE receives one or more Upload requests before performing a previously requested upload, the CPE MUST queue all requested uploads and perform each of them as closely as possible to the requested time (based on the value of the DelaySeconds argument and the time of the request). Queued uploads MUST be retained across reboots of the CPE. The CPE MUST be able to queue a minimum of three file transfers (downloads and uploads).

For each upload performed, the CPE MUST send a distinct TransferComplete. Note that the order in which a series of requested uploads will be performed might differ from the order of the corresponding requests due to differing values of DelaySeconds. For example, an ACS could request an upload with DelaySeconds equal to one hour, then five minutes later request a second upload with DelaySeconds equal to one minute. In this case, the CPE would perform the second upload before the first.

Table 56 – Upload arguments

Argument	Type	Description
`CommandKey`	string(32)	The string the CPE uses to refer to a particular upload. This argument is referenced in the methods Inform, TransferComplete, GetQueuedTransfers, GetAllQueuedTransfers and CancelTransfer. The value of the CommandKey is entirely at the discretion of the ACS and MAY be an empty string.

Argument	Type	Description
FileType	string(64)	An integer followed by a space followed by the file type description. Only the following values are currently defined for the FileType argument: “1 Vendor Configuration File” [DEPRECATED] “2 Vendor Log File” [DEPRECATED] “3 Vendor Configuration File <i>” “4 Vendor Log File <i>” For “3 Vendor Configuration File <i>”, <i> is replaced by the Instance Number from the Vendor Config File object as defined in the appropriate Root Data Model. The CPE uploads the file that corresponds to that entry in the vendor config file table. For “4 Vendor Log File <i>”, <i> is replaced by the Instance Number from the Vendor Log File object as defined in the appropriate Root Data Model. The CPE uploads the file that corresponds to that entry in the vendor log file table. The following format is defined to allow the unique definition of vendor-specific file types: "X <VENDOR> <Vendor-specific identifier>" <VENDOR> is replaced by a unique vendor identifier, which MAY be either an OUI or a domain name. The OUI or domain name used for a given vendor-specific file type MUST be one that is assigned to the organization that defined this method (which is not necessarily the same as the vendor of the CPE or ACS). An OUI is an organizationally unique identifier as defined in [13], which MUST be formatted as a 6 hexadecimal-digit OUI (organizationally unique identifier), with all upper-case letters and any leading zeros included. A domain name MUST be upper case with each dot (“.”) replaced with a hyphen or underscore. The FileType argument is intended to fully identify the file to be uploaded. If the standard values listed above are insufficient to uniquely identify the file, then vendor-specific file types MAY be used that provide more specific information to allow the intended file to be identified.
URL	string(256)	URL, as defined in [15], specifying the destination file location. HTTP and HTTPS transports MUST be supported. Other optional transports, as specified in Section 2.3.2, MAY be supported. When performing an upload to the URL specified by this argument, the CPE MUST make use of the HTTP PUT method. This argument specifies only the destination file location, and does not indicate in any way the name or location of the local file to be uploaded. The local file to be uploaded MUST be determined only by the FileType argument. This URL MUST NOT include the “userinfo” component, as defined in [15].
Username	string(256)	Username to be used by the CPE to authenticate with the file server. This string is set to the empty string if no authentication is required.
Password	string(256)	Password to be used by the CPE to authenticate with the file server. This string is set to the empty string if no authentication is required.
DelaySeconds	unsignedInt	The number of seconds from the time this method is called to the time the CPE is requested to initiate the upload. A value of zero indicates that no delay is requested. If a non-zero delay is requested, the upload MUST NOT occur in the same Session in which the request was issued. The CPE MUST perform the upload immediately after the time indicated by DelaySeconds, unless this is not possible for reasons outside the CPE’s control, in which case the CPE MUST attempt to perform the upload within one hour after the time indicated by DelaySeconds. If the CPE cannot begin the upload within this time window, the CPE MUST consider the upload to have failed and report this failure to the ACS using the TransferComplete method. The CPE MUST attempt to perform the upload within the time window specified above even if the CPE reboots one or more times prior to that time.

Table 57 – UploadResponse arguments

Argument	Type	Description
`Status`	int[0:1]	A successful response to this method returns an integer enumeration defined as follows: 0 = Upload has completed. 1 = Upload has not yet completed (for example, if the upload needs to wait until after the Session has been terminated). If the value of this argument is non-zero, the CPE MUST subsequently call the TransferComplete method to indicate the completion status of this upload (either successful or unsuccessful) either later in the same Session or in a subsequent Session.
`StartTime`	dateTime	The date and time upload was started in UTC. This need only be filled in if the upload has been completed. Otherwise, the value MUST be set to the Unknown Time value.
`CompleteTime`	dateTime	The date and time the upload was fully completed and applied in UTC. This need only be filled in if the upload has been completed. Otherwise, the value MUST be set to the Unknown Time value.

The following fault codes are defined for this method: 9000, 9001, 9002, 9003, 9004, 9011, 9012, 9013.

If an attempt is made to queue an upload when the file transfer queue is already full, the CPE MUST respond with fault 9004 (Resources exceeded). If the CPE detects the presence of the “userinfo” component in the file destination URL, it SHOULD reject the Upload request with the fault code 9003 (Invalid arguments).

A.4.1.6 FactoryReset

This method resets the CPE to its factory default state, and calls for use with extreme caution. The CPE MUST initiate the factory reset procedure only after successful completion of the Session. The calling arguments for this method are defined in Table 58. The arguments in the response are defined in Table 59.

Table 58 – FactoryReset arguments

Argument	Type	Description
-	void	This method has no arguments.

Table 59 – FactoryResetResponse arguments

Argument	Type	Description
-	void	This method response has no arguments.

The following fault codes are defined for this method: 9000, 9001, 9002, 9003.

A.4.1.7 GetAllQueuedTransfers

This method MAY be used by an ACS to determine the status of all queued downloads and uploads, including any that were not specifically requested by the ACS, i.e. autonomous transfers. The calling arguments for this method are defined in Table 60. The arguments in the response are defined in Table 61.

Table 60 – GetAllQueuedTransfers arguments

Argument	Type	Description
-	void	This method has no calling arguments.

Table 61 – GetAllQueuedTransfersResponse arguments

Argument	Type	Description
`TransferList`	AllQueuedTransferStruct[16]	Array of structures as defined in Table 62, each describing the state of one transfer that has not yet been fully completed.

Table 62 – AllQueuedTransferStruct definition

Name	Type	Description
`CommandKey`	string(32)	Set to the value of the CommandKey argument passed to CPE in the Download, ScheduleDownload or Upload method call that initiated the transfer, or an empty string for an autonomous transfer.
`State`	int[1:3]	The current state of the transfer. Defined values are: 1 = Not yet started 2 = In progress 3 = Completed, finishing cleanup All other values are reserved.
`IsDownload`	boolean	Indicates whether the transfer is a download (true) or an upload (false).
`FileType`	string(64)	An integer followed by a space followed by the file type description. Only the following values are currently defined for the FileType argument: "1 Firmware Upgrade Image" (download only) "2 Web Content" (download only) “3 Vendor Configuration File” (download or upload) [DEPRECATED for upload] “4 Vendor Log File” (upload only) [DEPRECATED] “4 Tone File” (download only; see [28] Appendix B) “5 Ringer File” (download only; see [28] Appendix B) “6 Vendor Configuration File <i>” (upload only) “7 Vendor Log File <i>” (upload only) “8 Stored Firmware Image” (download only) For “6 Vendor Configuration File <i>”, <i> is replaced by the Instance Number from the Vendor Config File object as defined in the appropriate Root Data Model. The Instance Number corresponds to that of the entry in the vendor config file that the CPE had been instructed to upload. For “7 Vendor Log File <i>”, <i> is replaced by the Instance Number from the Vendor Log File object as defined in the appropriate Root Data Model. The Instance Number corresponds to that of the entry in the vendor log file table that the CPE had been instructed to upload. The following format is defined to allow the unique definition of vendor-specific file types: "X <VENDOR> <Vendor-specific identifier>" <VENDOR> is replaced by a unique vendor identifier, which MAY be either an OUI or a domain name. The OUI or domain name used for a given vendor-specific file type MUST be one that is assigned to the organization that defined this method (which is not necessarily the same as the vendor of the CPE or ACS). An OUI is an organizationally unique identifier as defined in [13], which MUST be formatted as a 6 hexadecimal-digit OUI (organizationally unique identifier), with all upper-case letters and any leading zeros included. A domain name MUST be upper case with each dot (“.”) replaced with a hyphen or underscore.
`FileSize`	unsignedInt	The size of the file in bytes, or zero if this information is not available or if the CPE chooses not to make it available.

Name	Type	Description
`TargetFileName`	string(256)	The name of the file on the target (CPE) file system, or an empty string if this information is not available or if the CPE chooses not to make it available.

The following fault codes are defined for this method: 9000, 9001, 9002.

A.4.1.8 ScheduleDownload

Note – the functionality provided by this method overlaps that of the Download method [Section A.3.2.8]. Unlike Download, this method provides fine-grained control over when the download can be performed and applied. Also, this method does not permit a file to be downloaded and applied within the same Session.

This method MAY be used by the ACS to cause the CPE to download a specified file from the designated location and apply it within either one or two specified time windows. The CPE MUST support two time windows. The calling arguments for this method are defined in Table 63. The arguments in the response are defined in Table 64.

When a download is initiated using this method, the CPE MUST indicate successful or unsuccessful completion of the download via a TransferComplete message sent in a subsequent Session.

The CPE MUST only indicate successful completion of the download after the downloaded file(s) has been both successfully transferred and applied. While the criterion used to determine when a file has been successfully applied is specific to the CPE's implementation, the CPE SHOULD consider a downloaded file to be successfully applied only after the file is installed and in use as intended.

In the particular case of downloading a software image, as indicated by a "1 Firmware Upgrade Image" FileType, the CPE MUST consider the downloaded file(s) to be successfully applied only after the new software image is actually installed and operational. If the software image replaces the overall software of the CPE (which would typically require a reboot to install and begin execution), the software version represented in the Data Model MUST already reflect the updated software image in the Session in which the CPE sends a TransferComplete indicating successful download.

In the particular case of downloading a software image via the "6 Stored Firmware Image" FileType, the CPE MUST consider the downloaded file(s) to be successfully applied when they are downloaded, validated, and installed only. That is, the new firmware image does not actually have to be running in order for the device to consider the download to be completed.

If the file cannot be successfully downloaded or applied within the boundaries of the specified time windows, the CPE MUST NOT attempt to retry the file download on its own initiative, but instead MUST report the failure of the download to the ACS. Upon the ACS being informed of the failure of a download, the ACS MAY subsequently attempt to reinitiate the download by issuing a new ScheduleDownload request.

If an unrecoverable error occurs during a download, e.g. the file is not accessible or is corrupted, the file transfer MUST be aborted, even if the failure occurred on the first of two time windows.

If the CPE receives one or more Download or ScheduleDownload requests before performing a previously requested download, the CPE MUST queue all requested downloads and perform each of them as closely as possible to the requested time (based on the values of WindowStart in the time windows and the time of the request). Queued downloads MUST be retained across reboots and firmware upgrades of the CPE. The CPE MUST be able to queue a minimum of three file transfers (downloads and uploads).

For each download performed, the CPE MUST send a distinct TransferComplete. Note that the order in which a series of requested downloads will be performed might differ from the order of the corresponding requests due to differing time windows. For example, an ACS could request a download with a time window starting in one hour, then five minutes later request a second download with a time window starting in one minute. In this case, the CPE would perform the second download before the first.

All modifications to a CPE's configuration resulting from use of the ScheduleDownload method MUST be retained across reboots of the CPE.

If (and only if) the file transfer does not impact subscriber services, a CPE MAY transfer the file outside of a time window. For example, this might be the case for CPE which use Multicast streams for downloads. However, the CPE MUST never apply a downloaded file outside of a time window.

Table 63 – ScheduleDownload arguments

Argument	Type	Description
`CommandKey`	string(32)	The string the CPE uses to refer to a particular download. This argument is referenced in the methods Inform, TransferComplete, GetQueuedTransfers, GetAllQueuedTransfers and CancelTransfer. The value of the CommandKey is entirely at the discretion of the ACS and MAY be an empty string.

Argument	Type	Description
`FileType`	string(64)	An integer followed by a space followed by the file type description. Only the following values are currently defined for the FileType argument: "1 Firmware Upgrade Image" "2 Web Content" “3 Vendor Configuration File” “4 Tone File” (see [28] Appendix B) “5 Ringer File” (see [28] Appendix B) “6 Stored Firmware Image” (see Appendix V) The following format is defined to allow the unique definition of vendor-specific file types: "X <VENDOR> <Vendor-specific identifier>" <VENDOR> is replaced by a unique vendor identifier, which MAY be either an OUI or a domain name. The OUI or domain name used for a given vendor-specific file type MUST be one that is assigned to the organization that defined this method (which is not necessarily the same as the vendor of the CPE or ACS). An OUI is an organizationally unique identifier as defined in [13], which MUST be formatted as a 6 hexadecimal-digit OUI (organizationally unique identifier), with all upper-case letters and any leading zeros included. A domain name MUST be upper case with each dot (“.”) replaced with a hyphen or underscore. If and only if the CPE supports downloading of firmware images using the ScheduleDownload method, the CPE MUST support the "1 Firmware Upgrade Image" FileType value. The definition of a firmware upgrade image MAY vary across different CPE vendors, ranging from a single monolithic image to a set of inter-dependent files, but MUST be presented as a single URL to the CPE. For example, a URL could be a file that contains multiple URLs and instructions on how to upgrade the firmware image. All other FileType values are OPTIONAL. The FileType value of "2 Web Content" is intended to be used for downloading files that contain only web content for a CPE’s web-based user interface. A CPE that supports a web-based user interface and allows the content to be downloaded from the ACS via the ScheduleDownload method as a distinct file containing only web content SHOULD use the FileType value of "2 Web Content" when performing such a download. A CPE that supports a web-based user interface and allows the content to be downloaded from the ACS MAY instead include web content as part of its firmware upgrade image, or use some other means to update the web content in the CPE. Such a CPE need not support the FileType value of "2 Web Content". The FileType value of “3 Vendor Configuration File” is intended to be used for downloading a single vendor configuration file. A CPE MAY instead include one or more vendor configuration files as part of its firmware upgrade image.
`URL`	string(256)	URL, as defined in [15], specifying the source file location. HTTP and HTTPS transports MUST be supported. Other optional transports, as specified in Section 2.3.2, MAY be supported. If the CPE receives multiple ScheduleDownload requests with the same source URL, the CPE MUST perform each download as requested, and MUST NOT assume that the content of the file to be downloaded is the same each time. This URL MUST NOT include the “userinfo” component, as defined in [15].
`Username`	string(256)	Username to be used by the CPE to authenticate with the file server. This string is set to the empty string if no authentication is required.
`Password`	string(256)	Password to be used by the CPE to authenticate with the file server. This string is set to the empty string if no authentication is required.

Argument	Type	Description
FileSize	unsignedInt	The size of the file to be downloaded in bytes. The FileSize argument is intended as a hint to the CPE, which the CPE MAY use to determine if it has sufficient space for the file to be downloaded, or to prepare space to accept the file. The ACS MAY set this value to zero. The CPE MUST interpret a zero value to mean that that the ACS has provided no information about the file size. In this case, the CPE MUST attempt to proceed with the download under the presumption that sufficient space is available, though during the course of download, the CPE might determine otherwise. The ACS SHOULD set the value of this Parameter to the exact size of the file to be downloaded. If the value is non-zero, the CPE MAY reject the ScheduleDownload request on the basis of insufficient space. If the CPE attempts to proceed with the download based on the value of this argument, but the actual file size differs from the value of this argument, this could result in a failure of the download. However, the CPE MUST NOT cause the download to fail solely because it determines that the value of this argument is inaccurate.
TargetFile-Name	string(256)	The name of the file to be used on the target file system. This argument MAY be left empty if the target file name can be extracted from the downloaded file itself, or from the URL argument, or if no target file name is needed. If this argument is specified, but the target file name is also indicated by another source (for example, if it is extracted from the downloaded file itself), this argument MUST be ignored. If the target file name is used, the downloaded file would replace any existing file of the same name (whether or not the CPE archives the replaced file is a local matter). If present, this Parameter is treated as an opaque string with no specific requirements for its format. That is, the TargetFileName value is to be interpreted based on the CPE's vendor-specific file naming conventions. Note that this specification does not preclude the use of a file naming convention in which the file's path can be specified as part of the file name.
TimeWindow-List	TimeWin-dowStr-uct[1:2]	This structure defines the time window(s) during which the CPE MUST perform and apply the download. As noted earlier, if a file transfer does not generate additional network traffic and does not impact subscriber services, the CPE is permitted to perform (but not apply) the download outside of a time window. A CPE MUST be able to accept a request with either one or two TimeWindowStruct elements. The time windows MUST NOT overlap, i.e. if there are two time windows, the second window's WindowStart value has to be greater than or equal to the first window's WindowEnd value.

Table 64 – ScheduleDownloadResponse arguments

Argument	Type	Description
-	void	This method response has no arguments.

Table 65 – TimeWindowStruct definition

Name	Type	Description
WindowStart	unsignedInt	Start of this time window as an offset in seconds after receiving the download request. An offset is used in order to avoid a dependence on absolute time.
WindowEnd	unsignedInt	End of this time window as an offset in seconds after receiving the download request. An offset is used in order to avoid a dependence on absolute time.

Name	Type	Description
`WindowMode`	string(64)	An integer followed by a space followed by the time window mode description. The following values are currently defined: “1 At Any Time” “2 Immediately” “3 When Idle” “4 Confirmation Needed” The following format is defined to allow for the unique definition of vendor-specific time window modes: “X <VENDOR> <Vendor specific identifier>” <VENDOR> is replaced by a unique vendor identifier, which MAY be either an OUI or a domain name. The OUI or domain name used for a given vendor-specific file type MUST be one that is assigned to the organization that defined this method (which is not necessarily the same as the vendor of the CPE or ACS). An OUI is an organizationally unique identifier as defined in [13], which MUST be formatted as a 6 hexadecimal-digit OUI (organizationally unique identifier), with all upper-case letters and any leading zeros included. A domain name MUST be upper case with each dot (“.”) replaced with a hyphen or underscore. WindowMode specifies when within this time window the CPE is permitted to perform and apply the download. As noted earlier, if a file transfer does not impact subscriber services, the CPE is permitted to perform (but not apply) the download outside of a time window. The CPE MUST support “1 At Any Time”. This means that the CPE MAY perform and apply a download at any time during the time window even if this results in interruption of service for the subscriber. The CPE MUST support “2 Immediately”. This means that the CPE MUST perform and apply a download immediately at the start of the time window even if this results in interruption of service for the subscriber. The CPE MUST support “3 When Idle”. This means that interruption of service from the subscriber standpoint MUST NOT occur during the time window. How the CPE determines this is outside the scope of this specification. The CPE MAY support “4 Confirmation Needed”. This means that the CPE MUST ask for and receive confirmation before performing and applying the download. It is outside the scope of this specification how the CPE asks for and receives this confirmation. If confirmation is not received, this time window MUST NOT be used.
`UserMessage`	string(256)	A message to the user of the CPE, to inform him about a download request. The CPE MAY use this message when seeking confirmation from the user, e.g. when WindowMode is “4 Confirmation Needed”. When there is no need for such a message, it SHOULD be empty and MUST be ignored.
`MaxRetries`	int	The maximum number of retries for downloading and/or applying the file before regarding the transfer as having failed. Refers only to this time window (each time window can specify its own value). A value of 0 means “No retries are permitted”. A value of -1 means “the CPE determines the number of retries”, i.e. that the CPE can use its own retry policy, not that it has to retry forever.

The following fault codes are defined for this method: 9000, 9001, 9002, 9003, 9004, 9010, 9013.

If an attempt is made to queue an additional download when the CPE’s file transfer queue is already full, the CPE MUST respond with fault 9004 (Resources exceeded). If the CPE detects the presence of the “userinfo” component in the file source URL, or detects overlapping or otherwise invalid time windows (including zero windows supplied, or unsupported time window modes), it SHOULD reject the ScheduleDownload request with the fault code 9003 (Invalid arguments). If the CPE rejects the ScheduleDownload request because the FileSize argument exceeds the available space on the device, it MUST use the Download Failure (9010) fault code.

A.4.1.9 CancelTransfer

This method MAY be used by the ACS to cause the CPE to cancel a file transfer initiated by an earlier Download, ScheduleDownload or Upload method call. The TransferComplete method is not called for a file transfer that has successfully been canceled. The calling arguments for this method are defined in Table 66. The arguments in the response are defined in Table 67.

Table 66 – CancelTransfer arguments

Name	Type	Description
`CommandKey`	string(32)	The command key that was provided in the original Download, Upload or ScheduleDownload RPC.

Table 67 – CancelTransferResponse arguments

Name	Type	Description
-	void	This method response has no arguments

The following fault codes are defined for this method: 9000, 9001, 9004, 9021.

The CPE might be unable to cancel an active transfer, e.g. the file might currently be being downloaded in an uninterruptible way, or the CPE might be just about to apply the downloaded file. In this case, the CPE MUST respond with fault 9021 (Cancelation of file transfer not permitted in current transfer state). If the ACS is planning to cancel transfers, it SHOULD use a unique command key for each transfer. However, if the command key matches more than one transfer, the CPE MUST attempt to cancel all the matching transfers, and MUST respond with fault 9021 (described above) if it is unable to cancel all of them, in which case it SHOULD cancel as many matching transfers as it can. It is not an error to specify an invalid command key.

A.4.1.10 ChangeDUState

Appendix II / TR-157 Amendment 3 [32] details a Theory of Operation for Software Module Management, including defining the implicit and explicit state transitions for a DU.

This method MAY be used by an ACS to trigger the explicit state transitions of Install, Update, and Uninstall for a Deployment Unit (DU), i.e. installing a new DU, updating an existing DU, or uninstalling an existing DU. The calling arguments for this method are defined in Table 68. The arguments in the response are defined in Table 69.

When a DU state change is initiated using this method the CPE MUST indicate successful or unsuccessful completion of the state change via the DUStateChangeComplete method sent in a subsequent Session or via a CWMP fault sent within the same Session.

The ChangeDUState method MUST include one or more DU operations within a single method call, where a DU operation is described by one of the three types of operation structures (OperationStruct) that are defined in Table 70. There MUST, however, be only one resultant DUStateChangeComplete method for each ChangeDUState method issued by the ACS, and the DUStateChangeComplete MUST contain at least one result for each operation, including both successful and unsuccessful operations. The CPE MAY apply the operations in any order it chooses, but it MUST report the results for each operation in the same order as they were sent in the request. If the ACS wants to effect multiple state transitions for the same DU, then it SHOULD utilize multiple ChangeDUState RPCs to do so.

Regardless of the order in which the operations are applied, the CPE MUST complete each operation within one hour. If the CPE is unable to do so, it MUST consider that specific operation in error and send the appropriate FaultStruct in the resulting DUStateChangeComplete method call.

The CPE MUST send the related DUStateChangeComplete RPC within 24 hours of responding to the ChangeDUState method. If the CPE has not been able to complete all of the operations within that 24 hour time window, it MUST consider the remaining operations in error and send the appropriate FaultStruct within the resulting DUStateChangeComplete RPC.

If the ACS sends a request that contains more operation structures than the CPE can handle, the CPE MAY respond with a “Resources exceeded” (9004) CWMP Fault. The CPE MUST, however, be able to accept a minimum of sixteen (16) operation structures within a single request without issuing a “Resources exceeded” (9004) CWMP Fault.

If a DU state change fails, the CPE MUST NOT attempt to retry the state change on its own initiative, but instead MUST report the failure of the operation to the ACS using the DUStateChangeComplete method. Upon the ACS being informed of operation failure the ACS MAY subsequently attempt to reinitiate the DU state change by issuing a new ChangeDUState request.

Each DU operation contains an argument called UUID, which enables an ACS to uniquely identify a DU across CPE. The UUID is also a part of the Deployment Unit table’s unique key, along with the version of the DU and the Execution Environment that the DU is installed against. The format of the UUID and rules for generating the UUID are defined in RFC 4122 [37]. Additional rules for generating the UUIDs for Software Module Management are defined in Annex H. If the rules defined in RFC 4122 and Annex H are adhered to, both an ACS and a CPE will generate an equivalent UUID.

All modifications to a CPE’s configuration resulting from use of the ChangeDUState method MUST be retained across reboots of the CPE.

Table 68 – ChangeDUState Arguments

Argument	Type	Description
`Operations`	OperationStruct	The set of DU-related operations to be performed. The argument can contain any combination of the various OperationStruct types.
`CommandKey`	string(32)	The string the CPE uses to refer to a particular ChangeDUState. This argument is referenced in the methods Inform and DUStateChangeComplete. The value of the CommandKey is entirely at the discretion of the ACS and MAY be an empty string.

Table 69 – ChangeDUStateResponse Arguments

Argument	Type	Description
-	void	This method response has no arguments.

Table 70 – OperationStruct Types

Name	Type	Description
`InstallOpStruct`	OperationStruct	This is a type of OperationStruct used to Install new DUs on an Execution Environment.
`UpdateOpStruct`	OperationStruct	This is a type of OperationStruct used to Update existing DUs on an Execution Environment.
`UninstallOpStruct`	OperationStruct	This is a type of OperationStruct used to Uninstall existing DUs from an Execution Environment.

The three OperationStruct types in this table correspond to the three different explicit actions defined in the State Diagram in Appendix II / TR-157 Amendment 3 [32]. These are the structures that are allowed to appear in the Operations argument of the ChangeDUState RPC.

Table 71 – InstallOpStruct Definition

Name	Type	Description
`URL`	string(1024)	The URL, as defined in RFC 3986 [15], that specifies the location of the DU to be installed. HTTP and HTTPS transports MUST be supported. Other optional transports, as specified in Section 2.3.2, MAY be supported. If the CPE receives multiple Install requests with the same source URL, the CPE MUST perform each Install as requested, and MUST NOT assume that the content of the file to be downloaded is the same each time. This URL MUST NOT include the “userinfo” component, as defined in RFC 3986 [15].
`UUID`	string(36)	The UUID (see RFC 4122 [37] and Annex H) of the DU to be installed. The ACS MAY send down an empty string in which case the CPE MUST generate the UUID based on the rules defined in RFC 4122 [37] and Annex H.
`Username`	string(256)	Username to be used by the CPE to authenticate with the file server, if authentication is required.
`Password`	string(256)	Password to be used by the CPE to authenticate with the file server, if authentication is required.
`ExecutionEnvRef`	string(256)	A reference to the Execution Environment upon which the DU is to be installed. This argument is the Path Name of the Execution Environment Object instance, including its Instance Identifier. The Path Name MUST end with a “.” (dot) after the Instance Identifier of the Object. If this string is either not provided or sent in as an empty string, the CPE MUST choose which Execution Environment to use.

Table 72 – UpdateOpStruct Definition

Name	Type	Description
`UUID`	string(36)	The UUID (see RFC 4122 [37] and Annex H) of the existing DU that is to be updated.
`Version`	string(32)	The Version indicates which version of the DU to update when there are multiple versions available. If there are multiple versions available, this argument MUST be specified.

Name	Type	Description
`URL`	string(1024)	The URL, as defined in RFC 3986 [15], that specifies the location of the update to be applied to the existing DU(s). HTTP and HTTPS transports MUST be supported. Other optional transports, as specified in Section 2.3.2, MAY be supported. If the CPE receives an Update request with the same source URL as a previous Update or Install, the CPE MUST perform each Update as requested, and MUST NOT assume that the content of the file to be downloaded is the same each time. This URL MUST NOT include the "userinfo" component, as defined in RFC 3986 [15].
`Username`	string(256)	Username to be used by the CPE to authenticate with the file server, if authentication is required.
`Password`	string(256)	Password to be used by the CPE to authenticate with the file server, if authentication is required.

The combination of the UUID and URL determine which DU(s) will be updated. There are four possibilities (NOTE: if the URL is empty then the Username and Password SHOULD also be empty):

- UUID populated, URL empty: The CPE MUST Update the DU with the matching UUID based on its internal URL (the CPE SHOULD use the credentials that were last used to Install or Update this DU)
- UUID empty, URL populated: The CPE MUST Update the DU that last used the URL at either Install or Update (i.e. matches the URL Parameter in the DeploymentUnit.`{i}`. table)
- UUID populated, URL populated: The CPE MUST Update the DU with the matching UUID and update its internal URL
- UUID empty, URL empty: The CPE MUST Update all DUs based on their internal URL (the CPE SHOULD use the credentials that were last used to Install or Update the DU)

Note that because this option [UUID empty, URL empty] is intended to update all DUs, the Version MUST NOT be specified. If the Version is specified, the CPE SHOULD consider this operation in fault using 9003 as the fault code.

Table 73 – UninstallOpStruct Definition

Name	Type	Description
`UUID`	string(36)	The UUID (see RFC 4122 [37] and Annex H) of the existing DU that is to be uninstalled.
`Version`	string(32)	The version of the DU to be uninstalled. If this argument is not provided or is an empty string, all versions of the DU with the corresponding UUID are uninstalled.
`ExecutionEnvRef`	string(256)	A reference to the Execution Environment that the DU is to be uninstalled from. This argument is the Path Name of the Execution Environment Object instance, including its Instance Identifier. The Path Name MUST end with a "." (dot) after the Instance Identifier of the Object. If this string is either not provided or sent in as an empty string, the CPE MUST uninstall this DU from all Execution Environments that it is installed on.

The following fault codes are defined for this method: 9000, 9001, 9002, and 9004. These are the fault codes for the RPC as a whole; there can also be faults reported against specific operations contained in the DUStateChangeComplete FaultStruct (see A.4.2.3 for more details regarding the faults related to the individual operations). Appendix II.5 / TR-157 Amendment 3 [32] provides a description of the Software Module Management faults.

If the ACS sends a request that contains more operation structures than the CPE can handle, the CPE MAY respond with a 9004 (Resources Exceeded) CWMP Fault. Note that this scenario is differentiated from the 9027 (System Resources Exceeded) fault described in A.4.2.3, in which the CPE does not have the resources to perform the install or update of the DU.

A.4.2 ACS Methods

The methods listed in this Section MAY optionally be supported on an ACS. Only a CPE can call these methods.

A.4.2.1 Kicked

Note – this method is DEPRECATED due to the deprecation of Annex D, which defined the usage of this RPC.

The CPE calls this method whenever the CPE is "kicked" as described in Annex D. The calling arguments for this method are defined in Table 74. The arguments in the response are defined in Table 75.

Table 74 – Kicked arguments

Argument	Type	Value
`Command`	string(32)	Generic argument that MAY be used by the ACS for identification or other purposes.
`Referer`	string(64)	The content of the "Referer" HTTP header sent to the CPE when it was kicked.
`Arg`	string(256)	Generic argument that MAY be used by the ACS for identification or other purposes.
`Next`	string(1024)	The URL the ACS SHOULD return in the method response under normal conditions.

Table 75 – KickedResponse arguments

Argument	Type	Value
`NextURL`	string(1024)	The next URL the user's browser SHOULD be redirected to. This URL MAY include CGI arguments (for example, to maintain session state). If the ACS wishes to send the user's browser to a page on the CPE device itself, only the path portion of the URL is returned as a result (e.g. "/security/index.html"). This allows the CPE to use its canonical hostname in the HTTP 302 response. Note that this would require the ACS to have previous knowledge of available URLs on the CPE device through some mechanism outside the scope of this specification.

If this method returns a fault, the CPE SHOULD redirect the browser to an error page resident on the CPE device.

The following fault codes are defined for this method: 8000, 8001, 8002, 8003, 8005.

A.4.2.2 RequestDownload

This method allows the CPE to request a file download from the ACS. On reception of this request, the ACS MAY call the `Download` method to initiate the download. The calling arguments for this method are defined in Table 76. The arguments in the response are defined in Table 77.

Table 76 – RequestDownload arguments

Argument	Type	Value
`FileType`	string(64)	This is the FileType being requested (see Table 33 for the list of allowed file types).

Argument	Type	Value
`FileTypeArg`	ArgStruct[16]	Array of zero or more additional arguments, where each argument is a structure of name-value pairs as defined in Table 78. The use of the additional arguments depend on the FileType specified. The following arguments are defined for each of the currently defined file types.

FileType	FileTypeArg Names
1 Firmware Upgrade	(none)
2 Web Content	"Version"
3 Vendor Configuration File	(none)
4 Tone File	(none) (see [28] Appendix B)
5 Ringer File	(none) (see [28] Appendix B)

Argument	Type	Value
		The "8 Stored Firmware Image" File Type is not included in this list as it is not applicable to the RequestDownload mechanism. If the ACS receives arguments that it does not understand, it MUST ignore the unknown arguments, but process the request using the arguments that it does understand.

Table 77 – RequestDownloadResponse arguments

Argument	Type	Description
-	void	This method response has no arguments.

Table 78 – ArgStruct definition

Name	Type	Description
`Name`	string(64)	Argument name.
`Value`	string(256)	Argument value.

The following fault codes are defined for this method: 8000, 8001, 8002, 8003, 8005.

A.4.2.3 DUStateChangeComplete

This method informs the ACS of the completion of an earlier requested ChangeDUState method call, including both successful and unsuccessful operations. This method MUST be called only after the CPE has completed any file transfers related to the ChangeDUState request and attempted all of the operations specified in the ChangeDUState request, or if the ChangeDUState request times out. If the ACS fails the DUStateChangeComplete method, the CPE MUST NOT regard the ACS as having been informed of the completion of the file transfer, and MUST attempt to call the method again, either in the current Session or in a new Session, subject to the event delivery rules of Section 3.7.1.5.

There MUST be exactly one DUStateChangeComplete method for each ChangeDUState method called. The DUStateChangeComplete method MUST contain the results, whether success or failure, for each of the requested operations in the ChangeDUstate request. The entries in the Results argument MUST be in the same order as in the requesting ChangeDUState method, although the order in which the CPE actually applies the changes is up to the CPE implementation. There are situations in which a single ChangeDUState operation affects multiple Deployment Units. In this case there MUST be an OpResultStruct entry for each affected DU contained within the Results argument.

The calling arguments for this method are defined in Table 79. The arguments in the response are defined in Table 82.

Table 79 – DUStateChangeComplete Arguments

Name	Type	Description
`Results`	OpResultStruct…	The results of Operations performed against DUs.
`CommandKey`	string(32)	The value of the CommandKey argument passed to the CPE in the corresponding ChangeDUState method call.

Table 80 – OpResultStruct Definition

Name	Type	Description
`UUID`	string(36)	The UUID as defined in RFC 4122 [37] of the DU that was affected. In the case of an Install, this will be the UUID of the DU that was created. In the case of an Update or Uninstall, it will be the existing UUID of the DU that was either updated or uninstalled.
`DeploymentUnitRef`	string(256)	A reference to the DU affected. In the case of an Install, this is the DU that was created. In the case of an Update, this is the DU that was updated. In the case of an Uninstall, this is the DU that was removed. The DU reference is a full Path Name of the DeploymentUnit Object instance, including its Instance Identifier. The Path Name MUST end with a "." (dot) after the Instance Identifier of the Object.
`Version`	string(32)	The version of the DU affected. This MUST match the Version Parameter contained within the instance of the DeploymentUnit that is contained within the DeploymentUnitRef argument. In the case of an Install, this will be the version of the DU created. In the case of an Update, it will be the updated version of the DU. In the case of an Uninstall, it will be the version of the uninstalled DU.
`CurrentState`	string	The current state of the affected DU. This state was attained either by completing a requested Operation in the ChangeDUState method or reflects the state of the DU after a failed attempt to change its state. The following values are defined: *** Installed**: The DU is in an Installed state due to one of the following: successful Install, successful Update, failed Update, or failed Uninstall. In the case of a failed Update or failed Uninstall the Fault argument will contain an explanation of the failure. *** Uninstalled**: The DU was successfully Uninstalled from the device. * **Failed**: The DU could not be installed in which case a DU instance MUST NOT be created in the Data Model.
`Resolved`	boolean	Whether or not the DU operation resolved all of its dependencies. In the case of an Uninstall, this value is meaningless and SHOULD be true.
`ExecutionUnitRefList`	string	A comma-separated list of the Execution Units related to the affected DU. Each Execution Unit (EU) in the list is a full Path Name of the ExecutionUnit Object instance, including its Instance Identifier. The Path Name MUST end with a "." (dot) after the Instance Identifier of the Object. In the case of an Install, this will be the list of EUs that were created as a result of the DU's installation. In the case an Update, this will be the list of all EUs currently associated with the updated DU, including those that were created through the initial DU installation and any updates that had already occurred but not including any EUs that no longer exist on the device because of this or previous updates. In the case of an Uninstall, this will be the list of the EUs removed from the device due to the DU being removed.

Name	Type	Description
`StartTime`	dateTime	The date and time the operation on the DU was started in UTC. The CPE SHOULD record this information and report it in this argument, but if this information is not available, the value of this argument MUST be set to the Unknown Time value, as defined in Section 3.2 / TR-106 [16].
`CompleteTime`	dateTime	The date and time the operation on the DU was fully completed and applied in UTC. The CPE SHOULD record this information and report it in this argument, but if this information is not available, the value of this argument MUST be set to the Unknown Time value, as defined in Section 3.2 / TR-106 [16].
`Fault`	FaultStruct	A FaultStruct as defined in Table 81. If the operation was successful, the FaultCode MUST be zero. Otherwise a non-zero FaultCode is specified along with a FaultString indicating the failure reason.

Table 81 – FaultStruct Definition

Name	Type	Description
`FaultCode`	unsignedInt	The numerical fault code as defined in Section A.5.1. In the case of a fault, allowed values are: 9001, 9003, 9012, 9013, 9015, 9016, 9017, 9018, 9022, 9023, 9024, 9025, 9026, 9027, 9028, 9029, 9030, 9031 and 9032. A value of 0 (zero) indicates no fault.
`FaultString`	string(256)	A human-readable text description of the fault. This field SHOULD be empty if the FaultCode equals 0 (zero).

Appendix II.5 / TR-157 Amendment 3 [32] provides a description of the Software Module Management faults. The following error conditions are some examples of how a CPE could fail a specific operation:

- If the CPE cannot complete the operation for some unknown reason, it SHOULD reject the operation with a 9001 (Request Denied) fault code.
- If the CPE detects the presence of the “userinfo” component in the file source URL, it SHOULD reject the operation with a 9003 (Invalid Arguments) fault code.
- If the CPE cannot find the Execution Environment specified in the Install operation, it SHOULD reject the operation with a 9023 (Unknown Execution Environment) fault code.
- If the CPE determines that the Deployment Unit being installed does not match either the Execution Environment specified or any Execution Environment on the device, it SHOULD reject the operation with a 9025 (Deployment Unit to Execution Environment Mismatch) fault code
- If the CPE determines that the Deployment Unit being updated does not match the type of Execution Environment that it was previously installed against, it SHOULD reject the operation with a 9025 (Deployment Unit to Execution Environment Mismatch) fault code.
- If the CPE detects that the Deployment Unit being installed already has the same version as one already installed on the same Execution Environment, it SHOULD reject the operation with a 9026 (Duplicate Deployment Unit) fault code.

- If the CPE detects that that there are no more system resources (disk space, memory, etc.) to perform the Install or Update of a Deployment Unit, it SHOULD reject the operation with a 9027 (System Resources Exceeded) fault code.
- If the CPE cannot find the Deployment Unit specified in the Update operation, it SHOULD reject the operation with a 9028 (Unknown Deployment Unit) fault code.
- If a requested operation attempts to alter the State of a Deployment Unit in a manner that conflicts with the Deployment Unit State Machine Diagram (Appendix II / TR-157 Amendment 3 [32]), the CPE SHOULD reject the operation with a 9029 (Invalid Deployment Unit State) fault code.
- If a requested operation attempts to Uninstall a DU that caused an EE to come into existence, where that EE has at least 1 installed DU or at least 1 child EE, then the CPE SHOULD reject the operation with a 9029 (Invalid Deployment Unit State) fault code.

Table 82 – DUStateChangeCompleteResponse Arguments

Argument	Type	Description
-	void	This method response has no arguments.

The following fault codes are defined for this method: 8000, 8001, 8002, 8003, 8004, 8005.

A.4.2.4 AutonomousDUStateChangeComplete

This method informs the ACS of the completion (successful or unsuccessful) of a DU state change that was not specifically requested via CWMP using the ChangeDUState RPC. When used, this method MUST be called only after the CPE has completed any file transfers and carried out all operations related to the Autonomous DU State Change.

This method MAY contain the results from multiple autonomous DU state changes; it is implementation specific how the CPE chooses to aggregate the autonomous DU state changes, although the CPE MUST notify the ACS of any autonomous DU state changes within 24 hours of the time the operations were completed by the CPE. The CPE SHOULD make every attempt to aggregate, as much as possible, the autonomous change notifications to the ACS in the interest of scalability.

If the ACS fails this method, the CPE MUST NOT regard the ACS as having been informed of the completion of the file transfer, and MUST attempt to call the method again, either in the current Session or in a new Session, subject to the event delivery rules of Section 3.7.1.5.

The calling arguments for this method are defined in Table 83. The arguments in the response are defined in Table 86.

Table 83 – AutonomousDUStateChangeComplete Arguments

Name	Type	Description
`Results`	AutonOpResultStruct…	The results of Autonomous Operations performed against DUs.

Table 84 – AutonOpResultStruct Definition

Name	Type	Description
`UUID`	string(36)	The UUID as defined in RFC 4122 [37] of the DU that was affected by the autonomous state change. In the case of an Install, this will be the UUID of the DU that was created. In the case of an Update or Uninstall, it will be the existing UUID of the DU that was either updated or uninstalled.
`DeploymentUnitRef`	string(256)	A reference to the DU affected by the autonomous state change. In the case of an Install, this is the DU that was created. In the case of an Update, this is the DU that was updated. In the case of an Uninstall, this is the DU that was removed. The DU reference is a full Path Name of the DeploymentUnit Object instance, including its Instance Identifier. The Path Name MUST end with a "." (dot) after the Instance Identifier of the Object.
`Version`	string(32)	The version of the DU that was affected by the autonomous state change. This MUST match the Version Parameter contained within the instance of the DeploymentUnit that is contained within the DeploymentUnitRef argument. In the case of an Install, this will be the version of the DU created. In the case of an Update, it will be the updated version of the DU. In the case of an Uninstall, it will be the version of the uninstalled DU
`CurrentState`	string	The current state of the affected DU. This state was attained either by completing an autonomous Operation or reflects the state of the DU after a failed attempt to autonomously change its state. The following values are defined: * **Installed**: The DU is in an Installed state due to one of the following: successful Install, successful Update, failed Update, or failed Uninstall. In the case of a failed Update or failed Uninstall the Fault argument will contain an explanation of the failure. * **Uninstalled**: The DU was successfully uninstalled from the device. * **Failed**: The DU could not be installed in which case the DU instance MUST NOT be created in the Data Model.
`Resolved`	boolean	Whether or not the autonomous DU operation resolved all of its dependencies. In the case of an Uninstall, this value is meaningless and SHOULD be true.
`ExecutionUnitRefList`	string	A comma-separated list of the Execution Units related to the affected DU. Each Execution Unit (EU) in the list is a full Path Name of the ExecutionUnit Object instance, including its Instance Identifier. The Path Name MUST end with a "." (dot) after the Instance Identifier of the Object. In the case of an Install, this will be the list of EUs that were created as a result of the DU's installation. In the case an Update, this will be the list of all EUs currently associated with the updated DU, including those that were created through the initial DU installation and any updates that had already occurred, but not including any EUs that no longer exist on the device because of this or previous updates. In the case of an Uninstall, this will be the list of the EUs removed from the device due to the DU Un-Installation.
`StartTime`	dateTime	The date and time the autonomous operation on the DU was started in UTC. The CPE SHOULD record this information and report it in this argument, but if this information is not available, the value of this argument MUST be set to the Unknown Time value, as defined in Section 3.2 / TR-106 [16].

Name	Type	Description
`CompleteTime`	dateTime	The date and time the autonomous operation on the DU was fully completed and applied in UTC. The CPE SHOULD record this information and report it in this argument, but if this information is not available, the value of this argument MUST be set to the Unknown Time value, as defined in Section 3.2 / TR-106 [16].
`Fault`	FaultStruct	A FaultStruct as defined in Table 85. If the autonomous operation was successful, the FaultCode MUST be zero. Otherwise a non-zero FaultCode is specified along with a FaultString indicating the failure reason.
`OperationPerformed`	string	The operation that was performed against the DU via the autonomous state change. The following values are defined: **Install** – The autonomous Operation attempted was the Installation of a DU. **Update** – The autonomous Operation attempted was the Update of an existing DU. **Uninstall** – The autonomous Operation attempted was the Un-Installation of an existing DU.

Table 85 – FaultStruct Definition

Name	Type	Description
`FaultCode`	unsignedInt	The numerical fault code as defined in Section A.5.1. In the case of a fault, allowed values are: 9001, 9003, 9012, 9013, 9015, 9016, 9017, 9018, 9022, 9023, 9024, 9025, 9026, 9027, 9028, 9029, 9030, 9031 and 9032. A value of 0 (zero) indicates no fault.
`FaultString`	string(256)	A human-readable text description of the fault. This field SHOULD be empty if the FaultCode equals 0 (zero).

Appendix II.5 / TR-157 Amendment 3 [32] provides a description of the Software Module Management faults. The following error conditions are some examples of how a CPE could fail a specific operation:

- If the CPE cannot complete the operation for some unknown reason, it SHOULD reject the operation with a 9001 (Request Denied) fault code.
- If the CPE detects the presence of the “userinfo” component in the file source URL, it SHOULD reject the operation with a 9003 (Invalid Arguments) fault code.
- If the CPE cannot find the Execution Environment specified in the Install operation, it SHOULD reject the operation with a 9023 (Unknown Execution Environment) fault code.
- If the CPE determines that the Deployment Unit being installed does not match either the Execution Environment specified or any Execution Environment on the device, it SHOULD reject the operation with a 9025 (Deployment Unit to Execution Environment Mismatch) fault code
- If the CPE determines that the Deployment Unit being updated does not match the type of Execution Environment that it was previously installed against, it SHOULD reject the operation with a 9025 (Deployment Unit to Execution Environment Mismatch) fault code.

- If the CPE detects that the Deployment Unit being installed already has the same version as one already installed on the same Execution Environment, it SHOULD reject the operation with a 9026 (Duplicate Deployment Unit) fault code.
- If the CPE detects that that there are no more system resources (disk space, memory, etc.) to perform the Install or Update of a Deployment Unit, it SHOULD reject the operation with a 9027 (System Resources Exceeded) fault code.
- If the CPE cannot find the Deployment Unit specified in the Update operation, it SHOULD reject the operation with a 9028 (Unknown Deployment Unit) fault code.
- If a requested operation attempts to alter the State of a Deployment Unit in a manner that conflicts with the Deployment Unit State Machine Diagram (Appendix II / TR-157 Amendment 3 [32]), the CPE SHOULD reject the operation with a 9029 (Invalid Deployment Unit State) fault code.
- If a requested operation attempts to Uninstall a DU that caused an EE to come into existence, where that EE has at least 1 installed DU or at least 1 child EE, then the CPE SHOULD reject the operation with a 9029 (Invalid Deployment Unit State) fault code.

Table 86 – AutonomousDUStateChangeCompleteResponse Arguments

Argument	Type	Description
-	void	This method response has no arguments.

The following fault codes are defined for this method: 8000, 8001, 8002, 8003, 8004, 8005.

A.5 Fault Handling

A.5.1 CPE Fault Codes

Table 87 lists the fault codes that can be returned by a CPE. Note that the fault code values are shown in decimal representation.

Table 87 – Fault codes

Fault code	Description	Type[26]
9000	Method not supported	Server
9001	Request denied (no reason specified)	Server
9002	Internal error	Server
9003	Invalid arguments	Client
9004	Resources exceeded (when used in association with SetParameterValues, this MUST NOT be used to indicate Parameters in error)	Server

[26] The specified Type MUST be used to determine the value of the SOAP faultcode element as described in Section 3.4.7.

Fault code	Description	Type[26]
9005	Invalid Parameter name (associated with Set/GetParameterValues, GetParameterNames, Set/GetParameterAttributes, AddObject, and DeleteObject)	Client
9006	Invalid Parameter type (associated with SetParameterValues)	Client
9007	Invalid Parameter value (associated with SetParameterValues)	Client
9008	Attempt to set a non-writable Parameter (associated with SetParameterValues)	Client
9009	Notification request rejected (associated with SetParameterAttributes method).	Server
9010	File transfer failure (associated with Download, ScheduleDownload, TransferComplete or AutonomousTransferComplete methods).	Server
9011	Upload failure (associated with Upload, TransferComplete or AutonomousTrans-ferComplete methods).	Server
9012	File transfer server authentication failure (associated with Upload, Download, TransferComplete, AutonomousTransferComplete, DUStateChangeComplete, or AutonomousDUStateChangeComplete methods).	Server
9013	Unsupported protocol for file transfer (associated with Upload, Download, ScheduleDownload, DUStateChangeComplete, or AutonomousDUStateChangeComplete methods).	Server
9014	File transfer failure: unable to join multicast group (associated with Download, TransferComplete or AutonomousTransferComplete methods).	Server
9015	File transfer failure: unable to contact file server (associated with Download, TransferComplete, AutonomousTransferComplete, DUStateChangeComplete, or AutonomousDUStateChangeComplete methods).	Server
9016	File transfer failure: unable to access file (associated with Download, TransferComplete, AutonomousTransferComplete, DUStateChangeComplete, or AutonomousDUStateChangeComplete methods).	Server
9017	File transfer failure: unable to complete download (associated with Download, TransferComplete, AutonomousTransferComplete, DUStateChangeComplete, or AutonomousDUStateChangeComplete methods).	Server
9018	File transfer failure: file corrupted or otherwise unusable (associated with Download, TransferComplete, AutonomousTransferComplete, DUStateChangeComplete, or AutonomousDUStateChangeComplete methods).	Server
9019	File transfer failure: file authentication failure (associated with Download, TransferComplete or AutonomousTransferComplete methods).	Server
9020	File transfer failure: unable to complete download within specified time windows (associated with TransferComplete method).	Client
9021	Cancelation of file transfer not permitted in current transfer state (associated with CancelTransfer method).	Client
9022	Invalid UUID Format (associated with DUStateChangeComplete or AutonomousDUStateChangeComplete methods: Install, Update, and Uninstall)	Server
9023	Unknown Execution Environment (associated with DUStateChangeComplete or AutonomousDUStateChangeComplete methods: Install only)	Server
9024	Disabled Execution Environment (associated with DUStateChangeComplete or AutonomousDUStateChangeComplete methods: Install, Update, and Uninstall)	Server
9025	Deployment Unit to Execution Environment Mismatch (associated with DUStateChangeComplete or AutonomousDUStateChangeComplete methods: Install and Update)	Server
9026	Duplicate Deployment Unit (associated with DUStateChangeComplete or AutonomousDUStateChangeComplete methods: Install only)	Server
9027	System Resources Exceeded (associated with DUStateChangeComplete or AutonomousDUStateChangeComplete methods: Install and Update)	Server
9028	Unknown Deployment Unit (associated with DUStateChangeComplete or AutonomousDUStateChangeComplete methods: Update and Uninstall)	Server
9029	Invalid Deployment Unit State (associated with DUStateChangeComplete or AutonomousDUStateChangeComplete methods: Install, Update and Uninstall)	Server

Fault code	Description	Type[26]
9030	Invalid Deployment Unit Update – Downgrade not permitted (associated with DUStateChangeComplete or AutonomousDUStateChangeComplete methods: Update only)	Server
9031	Invalid Deployment Unit Update – Version not specified (associated with DUStateChangeComplete or AutonomousDUStateChangeComplete methods: Update only)	Server
9032	Invalid Deployment Unit Update – Version already exists (associated with DUStateChangeComplete or AutonomousDUStateChangeComplete methods: Update only)	Server
9800 – 9899	Vendor defined fault codes	-

A.5.2 ACS Fault Codes

Table 88 lists the fault codes that can be returned by an ACS. Note that the fault code values are shown in decimal representation.

Table 88 – Fault codes

Fault code	Description	Type[26]
8000	Method not supported	Server
8001	Request denied (no reason specified)	Server
8002	Internal error	Server
8003	Invalid arguments	Client
8004	Resources exceeded	Server
8005	Retry request	Server
8006	ACS version incompatible with CPE	Server
8800 – 8899	Vendor defined fault codes	-

A.6 RPC Method XML Schema

The XML schema, which is the normative definition for all RPC methods defined for the CPE WAN Management Protocol, is specified in the referenced files below:

Table 89 – CWMP XML Schema Versions

CWMP Version	Namespace	XSD
1.0	urn:dslforum-org:cwmp-1-0	http://www.broadband-forum.org/cwmp.php/cwmp-1-0.xsd
1.1	urn:dslforum-org:cwmp-1-1	http://www.broadband-forum.org/cwmp.php/cwmp-1-1.xsd
1.2	urn:dslforum-org:cwmp-1-2	http://www.broadband-forum.org/cwmp.php/cwmp-1-2.xsd
1.3	urn:dslforum-org:cwmp-1-2	http://www.broadband-forum.org/cwmp.php/cwmp-1-3.xsd
1.4	urn:dslforum-org:cwmp-1-2	http://www.broadband-forum.org/cwmp.php/cwmp-1-4.xsd

1

Annex B. Removed

Annex Removed.

Annex C. Signed Vouchers

Note – the mechanism defined in this Annex is DEPRECATED in favor of the "Software Module Management mechanism" as described in Appendix II / TR-157 Amendment 3 [32].

C.1 Overview

The CPE WAN Management Protocol defines an optional mechanism for securely enabling or disabling optional CPE capabilities. Unlike Parameters, the Voucher mechanism provides an additional layer of security for optional capabilities that require secure tracking (such as those involving payment).

A Voucher is a digitally signed data structure that instructs a CPE to enable or disable a set of Options. An Option is any optional capability of a CPE. When an Option is enabled, the Voucher can specify various characteristics that determine under what conditions that Option persists.

C.2 Control of Options Using Vouchers

An Option can be disabled, enabled, or enabled with expiration. An Option that is enabled with no expiration stays enabled until the ACS explicitly disables it. An Option that is enabled with expiration stays enabled only for the duration specified in the Voucher. After the specified duration period, the CPE MUST disable the Option itself.

An Option can also be defined as either transferable or non-transferable. If not otherwise specified, an Option enabled by a Voucher is non-transferable. A non-transferable Option is automatically disabled if the CPE becomes associated with a different broadband service provider than was in use at the time the Option was enabled. A transferable Option is one that is maintained with the CPE regardless of any subsequent changes of service provider.

Each Voucher, which can contain instructions to enable or disable one or more Options, MUST be digitally signed using the XML-Signature format [18]. Before applying the instructions in the Voucher, a CPE MUST validate the signature and authenticate the signer.

A Voucher is specific to a single CPE and cannot be used on a CPE other than the one indicated in the Voucher. This ensures that the mechanism used to distribute Vouchers can be used to ensure that only those CPEs that have properly appropriated an Option can enabled that Option.

A CPE supporting the use of Vouchers MUST support a network time synchronization protocol such as NTP or SNTP to ensure access to accurate time and date information. Application of a received voucher by the CPE, or comparison of an existing voucher against its expiration date, SHOULD only occur once the CPE has established network time.

The following Voucher-related methods are defined in Annex A of this specification:

- **SetVouchers**: Allows an ACS to download a list of Vouchers to a CPE. Each Voucher MAY enable or disable the Options defined within that Voucher.
- **GetOptions**: Allows an ACS to query the state of any or all Options supported by the CPE.

C.3 Voucher Definition

The RPC method SetVouchers allows an ACS to enable, disable, or modify the state of one or more Options. The SetVouchers method takes as an argument an array of Vouchers. Each Voucher in the array is separately Base64 encoded.

Prior to Base64 encoding, each Voucher is a signed XML structure utilizing the XML-Signature format [18]. Each independently signed Voucher MAY include one or more Option specifications. Each Option specification is a structure that specifies the intended state for the specified Option.

The elements of the Option specification are defined in Table 90. An Option MAY contain additional XML elements specific to the particular Option. An example Option specification structure is shown in Figure 4. An example of an entire signed Voucher is shown in Figure 5. In this example, two separate Options are enabled in the same Voucher.

Table 90 – Option specification definition

Name	Type	Description
`VSerialNum`	string(64)	Unique serial number identifying the particular Voucher. For a given ACS, each new Voucher created MUST be assigned a distinct Voucher serial number. This value MUST be unique across all CPE managed by that ACS and all Vouchers issued to a given CPE at different times.
`DeviceId`	DeviceIdStruct	A structure that uniquely identifies the particular CPE for which the Voucher is to apply. This structure is defined in Table 91. On receipt of a Voucher, a CPE MUST ensure that the information in the device ID matches its actual identity. If not, it MUST ignore the Voucher and respond with a Request Denied fault.
`OptionIdent`	string(64)	Identifying name of the particular Option to be enabled or disabled.
`OptionDesc`	string(256)	Text description of the Option.
`StartDate`	dateTime	Optional element. The date and time in UTC that the Option is to be enabled (only meaningful if Mode = EnableWithExpiration or EnableWithoutExpiration). If this element is not present, or if the specified time has already passed, an Option to be enabled is enabled immediately.
`Duration`	unsignedInt	Required if Mode = EnableWithExpiration. For an Option enabled with expiration, this element specifies the duration the Option will remain enabled in units of DurationUnits. If a start date is specified, the duration is relative to that start date.
`DurationUnits`	string	Required if Mode = EnableWithExpiration. This element specifies the units in which the duration element is specified. The allowed values are: "Days" "Months"

Name	Type	Description
`Mode`	string	This element specifies whether the designated Option is to be enabled or disabled, and if enabled, whether or not an expiration is specified. The allowed values are: “Disable” “EnableWithExpiration “EnableWithoutExpiration
`Transferable`	boolean	Optional element. A value of true (1) indicates that the Option is considered transferable, meaning that Option is to remain enabled until any specified expiration date regardless of any changes in service provider. If this element is false (0) or not present, the Option is considered non-transferable, requiring the Option be disabled upon change in service provider, associated with any change to the ProvisioningCode as defined in [27], [34], and [35].

Table 91 – DeviceIdStruct definition

Name	Type	Description
`Manufacturer`	string(64)	The manufacturer of the device. This parameter is for display only and need not be checked as part of the validation.
`OUI`	string(6)	Organizationally unique identifier of the device manufacturer. Represented as a six hexadecimal-digit value using all upper-case letters and including any leading zeros. The value MUST be a valid OUI as defined in [13].
`ProductClass`	string(64)	Identifier of the class of product for which the serial number applies. That is, for a given manufacturer, this parameter is used to identify the product or class of product over which the SerialNumber parameter is unique.
`SerialNumber`	string(64)	Identifier of the particular device that is unique for the indicated class of product and manufacturer.

Figure 4 – Example Option specification

```
<dsig:Object xmlns="" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" Id="option0">
  <Option>
    <VSerialNum>987654321</VSerialNum>
    <DeviceId>
      <Manufacturer>Example</Manufacturer>
      <OUI>012345</OUI>
      <ProductClass>Gateway</ProductClass>
      <SerialNumber>123456789</SerialNumber>
    </DeviceId>
    <OptionIdent>Option Name</OptionIdent>
    <OptionDesc>Option Description</OptionDesc>
    <StartDate>20021025T12:06:34</StartDate>
    <Duration>280</Duration>
    <DurationUnits>Days</DurationUnits>
    <Mode>EnableWithExpiration</Mode>
  </Option>
</dsig:Object>
```

Figure 5 – Example signed Voucher

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
                              20010315"></CanonicalizationMethod>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-
                              sha1"></SignatureMethod>
    <Reference URI="#option0">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
                              20010315"></Transform>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
      <DigestValue>TUuSqr2utLtQM5tY2DB1jL3nV00=</DigestValue>
    </Reference>
    <Reference URI="#option1">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
                              20010315"></Transform>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
      <DigestValue>/YX1C/E6zNf0+w4lG66NeXGOQB0=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>
    KAMfqOSnmGH52qRVGLNFEEM4PPkRSmMUGr2D8E3vwwW280e1Bn5pwQ==
  </SignatureValue>
  <KeyInfo>
    <KeyValue>
      <DSAKeyValue>
        <P>
          /X9TgR11EilS30qcLuzk5/YRt1I870QAwx4/gLZRJmlFXUAiUftZPY1Y+r/F9bow9s
          ubVWzXgTuAHTRv8mZgt2uZUKWkn5/oBHsQIsJPu6nX/rfGG/g7V+fGqKYVDwT7g/bT
          xR7DAjVUE1oWkTL2dfOuK2HXKu/yIgMZndFIAcc=
        </P>
        <Q>l2BQjxUjC8yykrmCouuEC/BYHPU=</Q>
        <G>
          9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0HgmdRWVeOutRZT+ZxBxCBgLRJFn
          Ej6EwoFhO3zwkyjMim4TwWeotUfI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTx
          vqhRkImog9/hWuWfBpKLZl6Ae1UlZAFMO/7PSSo=
        </G>
        <Y>
          TBASA/mjLI8bc2KM7u9X6nHHvjmPgZtTBhr1/Fzs2AkdYCYMwyy+v+OXU7u5e18JuK
          G7/uolVhjXNSn6ZgObF+wuMoyP/OUmNbSkdN1aRXXHPRsW2CcG3vjfV+Csg/LP3zfD
          xDkImsC8LuKXht/g4+nksA/3icRQXWagQJU9pUQ=
        </Y>
      </DSAKeyValue>
    </KeyValue>
    <X509Data>
      <X509IssuerSerial>
        <X509IssuerName>
          EMAILADDRESS=name@example.com,CN=Example,OU=CMS,O=Example,L=San\20Jose,
                              ST=California,C=US
        </X509IssuerName>
        <X509SerialNumber>4</X509SerialNumber>
      </X509IssuerSerial>
      <X509SubjectName>
        CN=eng.bba.certs.example.com,OU=CMS,O=Example,L=San\20Jose,ST=CA,C=US
      </X509SubjectName>
      <X509Certificate>
MIIEUjCCA7ugAwIBAgIBBDANBgkqhkiG9w0BAQUFADCBhDELMAkGA1UEBhMCVVMxEzARBgNVBAgT
CkNhbGlmb3JuaWExETAPBgNVBAcTCFNhbiBKb3NlMQ4wDAYDVQQKEwUyV2lyZTEMMAoGA1UECxMD
Q01TMQ4wDAYDVQQDEwUyV2lyZTEfMB0GCSqGSIb3DQEJARYQZWJyb3duQDJ3aXJlLmNvbTAeFw0w
MjA5MDUyMDU4MTZaFw0xMjA5MDIyMDU4MTZaMG0xCzAJBgNVBAYTAlVTMQswCQYDVQQIEwJDQTER
MA8GA1UEBxMIU2FuIEpvc2UxDjAMBgNVBAoTBTJXaXJlMQwwCgYDVQQLEwNDTVMxIDAeBgNVBAMT
F2VuZy5iYmEuY2VydHMuMndpcmUuY29tMIIBtzCCASwGByqGSM44BAEwggEfAoGBAP1/U4EddRIp
Ut9KnC7s5Of2EbdSPO9EAMMeP4C2USZpRV1AIlH7WT2NWPq/xfW6MPbLm1Vs14E7gB00b/JmYLdr
mVClpJ+f6AR7ECLCT7up1/63xhv4O1fnxqimFQ8E+4P208UewwI1VBNaFpEy9nXzrith1yrv8iID
GZ3RSAHHAhUAl2BQjxUjC8yykrmCouuEC/BYHPUCgYEA9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC
+VdMCz0HgmdRWVeOutRZT+ZxBxCBgLRJFnEj6EwoFhO3zwkyjMim4TwWeotUfI0o4KOuHiuzpnWR
bqN/C/ohNWLx+2J6ASQ7zKTxvqhRkImog9/hWuWfBpKLZl6Ae1UlZAFMO/7PSSoDgYQAAoGATBAS
A/mjLI8bc2KM7u9X6nHHvjmPgZtTBhr1/Fzs2AkdYCYMwyy+v+OXU7u5e18JuKG7/uolVhjXNSn6
ZgObF+wuMoyP/OUmNbSkdN1aRXXHPRsW2CcG3vjfV+Csg/LP3zfDxDkImsC8LuKXht/g4+nksA/3
icRQXWagQJU9pUSjgdAwgc0wHQYDVR0OBBYEFMTl/ebdHLjaEoSS1PcLCAdFX32qMIGbBgNVHSME
gZMwgZChgYqkgYcwgYQxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMREwDwYDVQQH
EwhTYW4gSm9zZTEOMAwGA1UEChMFMldpcmUxDDAKBgNVBAsTA0NNUzEOMAwGA1UEAxMFMldpcmUx
HzAdBgkqhkiG9w0BCQEWEGVicm93bkAyd2lyZS5jb22CAQAwDgYDVR0PAQH/BAQDAgeAMA0GCSqG
SIb3DQEBBQUAA4GBAF1PGAbyvA0p+6o7nXfF3jzAdoHdaZh55C8sOQ9J62IF8D1jl6JxR7pjcCp2
iYmWkwQMncGfq+X8xP7BIqntDmIlYXuDTlXbyxXsu6lnT7nCbJwMwlLOxFwN+Axy7BM3NkAFE5Mb
aaoJWtmD1QrvcAFfDhLeBT+tIRueK7Pq9LDS
      </X509Certificate>
      <X509Certificate>
```

```
MIICeTCCAeICAQAwDQYJKoZIhvcNAQEEBQAwgYQxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxp
Zm9ybmlhMREwDwYDVQQHEwhTYW4gSm9zZTEOMAwGA1UEChMFMldpcmUxDDAKBgNVBAsTA0NNUzEO
MAwGA1UEAxMFMldpcmUxHzAdBgkqhkiG9w0BCQEWEGVicm93bkAyd2lyZS5jb20wHhcNMDEwNzMx
MDMwNjQ5WhcNMDcwMTIxMDMwNjQ5WjCBhDELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3Ju
aWExETAPBgNVBAcTCFNhbiBKb3NlMQ4wDAYDVQQKEwUyV2lyZTEMMAoGA1UECxMDQ01TMQ4wDAYD
VQQDEwUyV2lyZTEfMB0GCSqGSIb3DQEJARYQZWJyb3duQDJ3aXJlLmNvbTCBnzANBgkqhkiG9w0B
AQEFAAOBjQAwgYkCgYEA1ISJbL6i0J/6SBoet3aA8fki8s7pb/QUZueWj+0YKoDaQWh4MUCT0K06
N/0Z2cLMVg8JyezEpdnh3lVM/Ni5ow2Mst4dpdccQQEHouqwNUWIBFU196/LPRyLjoM2NeIXSKMj
AdPwvcenxmqeVBr/ZUmr4JQpdSI2AZJuHvCIjUsCAwEAATANBgkqhkiG9w0BAQQFAAOBgQBa3CCX
ga9L0qrGWxpNj312Az+tYz8bpEp2e2pAVrJHdW/CJ0uRlE341oTkhfYFa5CuuieF7Jcwf1B3+cGo
JrLWqeKqsNnrbmMFC/9hnrLlgZKEKi0POaGSFS/Pw9nodGWFZCiaQmeG+J6CWeASiFMdwgRGvESW
axfzzIKiXsXwkA==
      </X509Certificate>
    </X509Data>
  </KeyInfo>
  <dsig:Object xmlns="" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" Id="option0">
    <Option>
      <VSerialNum>987654321</VSerialNum>
      <DeviceId>
        <Manufacturer>Example</Manufacturer>
        <OUI>012345</OUI>
        <ProductClass>Gateway</ProductClass>
        <SerialNumber>123456789</SerialNumber>
      </DeviceId>
      <OptionIdent>First option name</OptionIdent>
      <OptionDesc>First option description</OptionDesc>
      <StartDate>20021025T12:06:34</StartDate>
      <Duration>280</Duration>
      <DurationUnits>Days</DurationUnits>
      <Mode>EnableWithExpiration</Mode>
    </Option>
  </dsig:Object>
  <dsig:Object xmlns="" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" Id="option1">
    <Option>
      <VSserialNum>987654322</VSerialNum>
      <DeviceId>
        <Manufacturer>Example</Manufacturer>
        <OUI>00D09E</OUI>
        <ProductClass>Gateway</ProductClass>
        <SerialNumber>123456789</SerialNumber>
      </DeviceId>
      <OptionIdent>Second option name</OptionIdent>
      <OptionDesc>Second option description</OptionDesc>
      <StartDate>20021025T12:06:34</StartDate>
      <Duration>280</Duration>
      <DurationUnits>Days</DurationUnits>
      <Mode>EnableWithExpiration</Mode>
    </Option>
  </dsig:Object>
</Signature>
```

Annex D. Web Identity Management

Note – the mechanism defined in this Annex is DEPRECATED due to exploits like CSRF (cross-site request forgery) and XSS (cross-site scripting), which turn the home network into an untrusted environment. JavaScript downloaded from the Internet could allow a malicious script to perform redirects and connect to a web site or portal with the "unknowing" subscriber web identity.

D.1 Overview

To support web-based applications or other CPE-related web pages on a back-end web site for access from a browser within the CPE's local network, the CPE WAN Management Protocol provides an optional mechanism that allows such web sites to customize their content with explicit knowledge of the customer associated with that CPE. That is, the location of users browsing from inside the CPE's LAN can be automatically identified without any manual login process.

The protocol defines a set of optional interfaces that allow the web site to initiate communication between the CPE and ACS, which allows a web site in communication with that ACS to identify which CPE the user is operating behind. This allows the web site to customize its content to be specific to the associated broadband account, the particular type of CPE, or any other characteristic that is known to the ACS.

Note – this identification mechanism does not distinguish among different users on the same network behind a single CPE. In situations where identification of a specific user is required, a separate identity management mechanism, such as manual login, would be needed.

D.2 Use of the Kicked RPC Method

The CPE WAN Management Protocol defines an optional Kicked RPC method in Annex A, which can be used to support web identity management functionality.

The CPE's invocation of the Kicked method is initiated by an external stimulus to the CPE. This external stimulus is assumed to be web-based, and thus the associated method provides a means to communicate information that would be useful in a web-based transaction. A suggested definition of the stimulus interface is given in Section D.4.

The information contained in the Kicked method call includes both the information needed to uniquely identify the CPE, but also parameters that can be used to associate the method call with a particular web browser session.

The response to the Kicked method allows the ACS to specify a URL to which the browser SHOULD be redirected. This URL MAY contain CGI arguments that allow the ACS to continue to track the browser session.

D.3 Web Identity Management Procedures

The Web Identity Management mechanism is based on a model in which a web server is associated with and can communicate with an ACS. Whenever this web server wishes to either identify the user's CPE or cause the CPE to establish communication with the ACS for some other purpose, the following sequence of events will occur (under normal conditions):

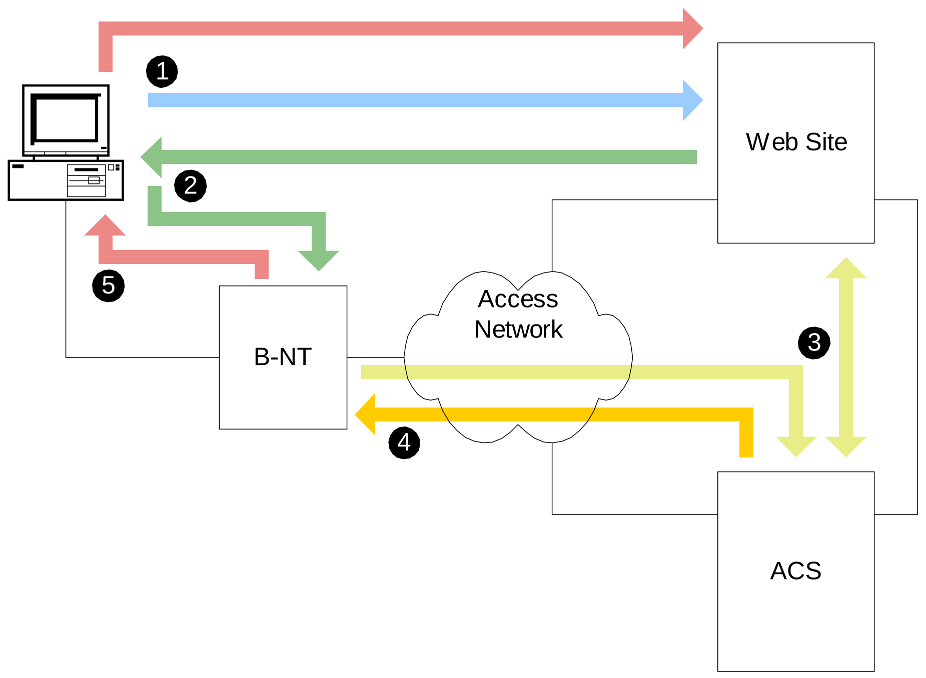
1. The user's browser accesses a web page that requires knowledge of, or communication with, the user's CPE.
2. The web site redirects the browser to a specific URL accessible only from the CPE's private-network (LAN) interface through which the browser "kicks" the CPE, providing the CPE via CGI arguments with information it needs to follow the subsequent steps (see Section D.4).
3. The CPE notifies the ACS that it has been kicked, using the "Kicked" RPC method call defined in Annex A. In this method call, the CPE identifies itself and passes information to uniquely identify the browser session.
4. The ACS responds to this method call by passing a URL that the CPE SHOULD redirect the user's browser. This URL would normally include CGI arguments that identify the session state. While the connection is open, the ACS MAY also initiate any other appropriate RPC transactions.
5. The CPE responds to the browser's HTTP request by redirecting the browser to the URL indicated by the ACS.

This exchange allows the ACS to uniquely identify the CPE; potentially generate a custom page based on knowledge of the particular user, their equipment, and any associated account privileges; and then direct the user to that customized page.

The ACS MAY also initiate any other RPC transactions that are appropriate given the particular user action. For example, if a user requests a firmware upgrade to their CPE from a web page, the ACS could instruct the CPE to initiate a file download over the same connection that the ACS responds to the Kicked method call.

Figure 6 shows the sequence of events associated with this mechanism. The numbers shown correspond to the step numbers above.

Figure 6 – Sequence of events for the "kick" mechanism



D.4 LAN Side Interface

A CPE MAY support web identity management by providing a LAN-side web URL accessible from a browser operating on the local network.

The associated web server in the CPE SHOULD support CGI arguments to be passed to corresponding arguments in the Kicked RPC method defined in Annex A. The RECOMMENDED arguments are listed in Table 92.

Table 92 – Recommended CGI Arguments for the kick URL

Name	Type	Value
command	string(32)	The value to be passed in the Command argument of the Kicked method call. This CGI argument allows the ACS to identify a command it is to perform in response to the resulting Kicked method call.
arg	string(256)	The value to be passed in the Arg argument of the Kicked method call. This CGI argument MAY be used by the ACS to pass arguments for the corresponding command. The particular uses for this argument are not defined.
next	string(1024)	The value to be passed in the Next argument of the Kicked method call. This contains the URL the web site wishes the browser be sent after the Kicked process has completed. The ACS processing the Kicked method MAY override this request and return a different URL in the Kicked response.

To initiate the kick process, the browser would be sent to the CPE's URL, for example via an HTTP 302 redirect or via a form post. This access would include the CGI arguments as defined in Table 92. For example, the browser might be redirected to:

```
http://cpe-host-
name/kick.html?command=<#>&arg=<arg>&next=<url>
```

After the CPE receives the corresponding HTTP GET request, the CPE SHOULD initiate a `Kicked` method call, using the CGI arguments to fill in the method arguments as defined in Annex A.

The CPE SHOULD limit the number of Kicked method calls it sends to the ACS per hour to a defined maximum value. Receiving a kick request that would result in exceeding this maximum value is considered a security violation and SHOULD NOT result in a call to the Kicked method.

Annex E. Signed Package Format

Note – the mechanism defined in this Annex is DEPRECATED in favor of the “Software Module Management mechanism” as described in Appendix II / TR-157 Amendment 3 [32].

E.1 Introduction

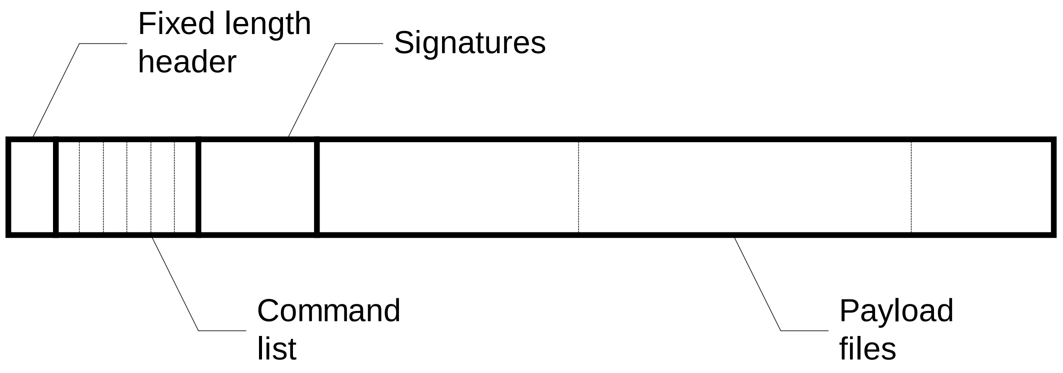
This document specifies a signed package format that MAY be used to securely download files into a recipient device. The format allows one or more files to be encapsulated within a single signed package. The package format allows the recipient to authenticate the source, and contains instructions for the recipient to extract and install the contents.

The signed package format is intended to be used for download from a server via HTTP, HTTPS, or FTP file transfer, or via other means of file transfer from a remote or local source.

E.2 Signed Package Format Structure

The basic format of a signed package file is shown in Figure 7.

Figure 7 – Signed package format



A general description of each of the signed package format components is given in Table 93.

Table 93 – Signed package component summary

Component	Description
Header	The header is a fixed-length structure including a preamble, format version, and the lengths of the command list and payload components.
Command list	The command list contains a sequence of instructions to be followed in extracting and installing the files contained within the package. Each command is in the form of a type-length-value (TLV).
Signatures	This section of the package contains a PKCS #7 digital signature block containing a set of zero or more digital signatures as described in Section E.5.

Component	Description
Payload files	This section of the package contains one or more files to be installed following the instructions in the command list. This document does not define any specific payload file formats.

E.2.1 Encoding Conventions

The following encoding conventions are used throughout this specification unless explicitly stated otherwise:

- Multi-octet numeric values are encoded in network byte order (big endian format).
- File or directory Path names are specified in UNIX format (e.g., "/dir/dir/base.ext").

E.3 Header Format

The signed package header is a fixed-length 24-octet structure. The format of the header is defined in Table 94.

Table 94 – Signed package header format

Field	Type	Description
Preamble	8 octets	A fixed sequence of octets containing the following hexadecimal values: 32 57 49 52 45 5F 53 50 An interpreter of the signed package format MUST verify that the preamble contains exactly this sequence of values for the package to be considered valid.
Major version	32-bit integer	Value indicating the major component of the package format version. An implementation conforming to this specification has a major version of 1 (one). Changes to the major version denote incompatible changes to this format.
Minor version	32-bit integer	Value indicating the minor component of the package format version. An implementation conforming to this specification has a minor version of 0 (zero). Changes to the minor version denote compatible changes to the package format. An implementation implementing this version of the specification SHOULD be capable of interpreting packages encoded using a format with a different minor version value.
Command list length	32-bit integer	Length in octets of the command list. The command list length MUST be less than 216.
Payload length	32-bit integer	Length in octets of the payload, including all files contained within it.

E.4 Command List Format

Each command in the command list has a format specified in Table 95.

Table 95 – Command format

Field	Type	Description
Type	32-bit integer	Specifies the particular command.
Length	32-bit integer	Specifies the length in octets of the Value field. The total length of the command is Length + 8 octets.
Value	(Conditional)	Zero or more octets of parameters associated with the particular command type.

If a recipient of this file format finds a Type value that is unknown to it, it MUST ignore the command and continue parsing the remainder of the package, using the Length value to skip to the next command, if any.

E.4.1 Command Types

The command list contains two types of commands: package parameters and actions to be taken. Examples of package parameters include the software version of a contained software image or a timeout for the remainder of the download. Examples of actions are add, remove, and move. The actions taken together in the order specified in the command list define the sequence of modifications to the file system required to extract and install the contained files.

The file-related commands have two variants: one that operates on explicit files and another that operates on versioned files. The name of a versioned file has a fixed "base" up to 8 characters in length, and an "extension" that is 3 characters in length. Each time the content of a versioned file is updated, the file extension is changed to a new value that indicates the file version. Because of this, if an upgrade needs to replace a versioned file, any existing file with the same base name but different extension MUST be removed.

The specific commands defined by this specification are listed in Table 96.

Table 96 – Command Type summary

Type	Command name
0	End
1	Extract File
2	Extract Versioned File
3	Add File
4	Add Versioned File
5	Remove File
6	Remove Versioned File
7	Remove Sub-Tree
8	Move File
9	Move Versioned File
10	Version
11	Description
12	Recoverable Timeout
13	Unrecoverable Timeout
14	Initial Timeout
15	Initial Activity Timeout
16	Reboot
17	Format File System
18	Minimum Version
19	Maximum Version
20	Role
21	Minimum Non-Volatile Storage
22	Minimum Volatile Storage Size
23	*Reserved*
24	*Reserved*
25	Required Attributes
1000-9999	Vendor-specific commands

E.4.2 End Command

This command signifies the end of the command list. This command need not be present in a command list, but if encountered a recipient MUST stop parsing the remainder of the command list portion of the package.

The Length parameter for this command MUST be 0 (zero), indicating that no Value field follows.

E.4.3 Extract and Add Commands

The extract and add commands include Extract File, Extract Versioned File, Add File, and Add Versioned File. The extract commands instruct the recipient to remove any existing file of the same name and replace it with the specified file in the payload.

The add commands instruct the recipient to first check for an existing file of the same name, and only install the new file if no existing file can be found.

For the versioned file variants of these commands, the above operations consider an existing file as any file that has the same base name as the specified file. That is, the Extract Versioned File command removes all existing files with the same base name and any extension prior to installing the new file. Similarly, the Add Versioned File command checks for any file with the same base name as the specified file, regardless of extension, and only installs the new file if no such file can be found.

When a new file is to be created in a directory that does not exist, the recipient MUST create the required directory.

All of the extract and add commands include information in the Value portion of the command. The format of this information is defined in Table 97.

Table 97 – Value format for the extract and add commands

Field	Type	Description
Flags	32-bit integer	A bit-field defined as follows: Bit 0 (LSB): Unsafe Flag. A 1 (one) value of this flag indicates that if this command completes successfully, but a subsequent command in the command list fails, the recipient device will be left in an unsafe state, and SHOULD follow its procedures for recovery of its file system to a known safe state. All other bits are reserved and MUST be set to 0 (zero) and MUST be ignored by the recipient.
Path Offset	32-bit integer	The offset in octets from the beginning of the Value field to the Path field in this command.
Path Length	32-bit integer	The length of the Path field in octets.
Hash Type	32-bit integer	Type of hash algorithm used in creating the Hash field. The following values are currently defined: 1 = SHA-1. When set to this value, the Hash field contains the 20-octet SHA-1 hash of the specified file. The Hash Length value in this case MUST be set to 20 (decimal). All other values are reserved.
Hash Offset	32-bit integer	The offset in octets from the beginning of the Value field to the Hash field in this command.
Hash Length	32-bit integer	The length of the Hash field in octets.
File Offset	32-bit integer	The offset in octets from the beginning of the payload portion of the package to the beginning of the specified file.

Field	Type	Description
File Length	32-bit integer	The length of the file payload in octets. The actual contents of the file are found in the file payload portion of the package.
Path	String of length *Path Length*	Path of the specified file, including the directory tree and file name.
Hash	Octet string of length *Hash Length*	Hash of the payload file using the hash algorithm defined in the Hash Type field. The hash of the payload file is included in the command because the signatures validate only the package header and command list. By including the file hash in the command, the signature ensures the validity of the file contents.

E.4.4 Remove Commands

The remove commands include Remove File, Remove Versioned File, and Remove Sub-Tree.

The Remove File command removes the file with the specified path, if it exists.

The Remove Versioned File command removes all files with the same base as the specified file, regardless of extension.

The Remove Sub-Tree command removes all files and directories beneath and including the specified path.

All of the remove commands include information in the Value portion of the command. The format of this information is defined in Table 98.

Table 98 – Value format for the remove commands

Field	Type	Description
Flags	32-bit integer	A bit-field defined as follows: Bit 0 (LSB): Unsafe Flag. A 1 (one) value of this flag indicates that if this command completes successfully, but a subsequent command in the command list fails, the recipient device will be left in an unsafe state, and SHOULD follow its procedures for recovery of its file system to a known safe state. All other bits are reserved and MUST be set to 0 (zero) and MUST be ignored by the recipient.
Path Offset	32-bit integer	The offset in octets from the beginning of the Value field to the Path field in this command.
Path Length	32-bit integer	The length of the Path field in octets.
Path	String of length *Path Length*	Path of the specified file or directory.

E.4.5 Move Commands

The move commands include Move File and Move Versioned File.

The Move File command renames a file to the name specified in this command. If the destination path specified indicates a different directory, the file is moved to the indicated destination directory.

The Move Versioned File command moves a file matching the base name of the file specified in the source path, regardless of the extension. If more than one such file exists in the specified directory, only one of the files is moved and the others are deleted. If the versioned file extension string is a decimal number, then the lowest numbered file is moved and the rest are deleted.

In all cases, if there is already a file with the same path as the specified destination file, the move commands will overwrite that file.

If the source file specified in a move command does not exist, no action is taken, and the recipient continues to process the remaining commands in the command list.

All of the move commands include information in the Value portion of the command. The format of this information is defined in Table 99.

Table 99 – Value format for the move commands

Field	Type	Description
Flags	32-bit integer	A bit-field defined as follows: Bit 0 (LSB): Unsafe Flag. A 1 (one) value of this flag indicates that if this command completes successfully, but a subsequent command in the command list fails, the recipient device will be left in an unsafe state, and SHOULD follow its procedures for recovery of its file system to a known safe state. All other bits are reserved and MUST be set to 0 (zero) and MUST be ignored by the recipient.
Source Path Offset	32-bit integer	The offset in octets from the beginning of the Value field to the Source Path field in this command.
Source Path Length	32-bit integer	The length of the Source Path field in octets.
Destination Path Offset	32-bit integer	The offset in octets from the beginning of the Value field to the Destination Path field in this command.
Destination Path Length	32-bit integer	The length of the Destination Path field in octets.
Source Path	String of length *Source Path Length*	Path of the source file.
Destination Path	String of length *Destination Path Length*	Path of the destination to which the source file is to be moved/renamed.

E.4.6 Version and Description Commands

The Value field for both the Version and Description commands contain a single UTF-8 string to be used for informational, display, or logging purposes.

The Version field is intended to indicate the overall version associated with the package. For example, if the package contains a software upgrade (which can include many individual files), the Version field MAY be used to indicate the new software version associated with the upgrade.

E.4.7 Timeout Commands

The timeout commands include Initial Timeout, Initial Activity Timeout, Recoverable Timeout, and Unrecoverable Timeout.

The timeout commands specify a timeout value for the continued download of the package file before the download SHOULD be terminated. These commands are to accommodate the case where the command and signature portions of the package are downloaded and interpreted prior to downloading the remainder of the package file. The timeout commands MAY be used to control the timeout parameters associated with a download process of this type. If the package is downloaded or received as a whole prior to interpreting the package contents, the timeout commands MAY be ignored.

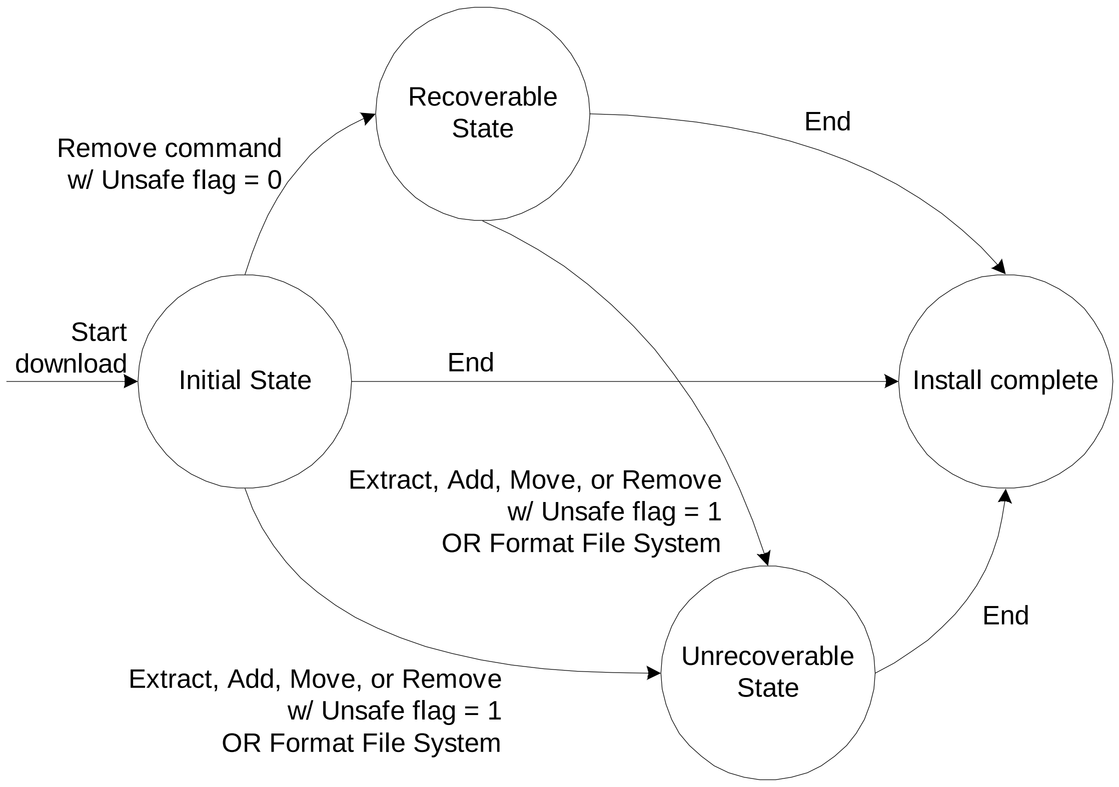
Each timeout command includes information in the Value portion of the command. The format of this information is defined in Table 100.

Table 100 – Value format for the timeout commands

Field	Type	Description
Timeout	32-bit Integer	The timeout value in seconds relative to the beginning of the package download operation. A value of 0 (zero) indicates an infinite timeout.

Each of the timeout commands allows a distinct timeout value to be specified, where the Timeout field in that command indicates the desired value. The use of each timeout value is based on the state of the recipient as it processes commands using the state transition model shown in Figure 8. The figure shows the state transitions that occur as each command in the command list is processed in sequence. For each command processed, the state remains the same until one of the cases indicated by the state transition arrows occurs.

Figure 8 – Download state diagram used for timeout model


Recoverable State
End
Remove command w/ Unsafe flag = 0
Start download
Initial State
End
Install complete
Extract, Add, Move, or Remove w/ Unsafe flag = 1 OR Format File System
End
Unrecoverable State
Extract, Add, Move, or Remove w/ Unsafe flag = 1 OR Format File System

The above state diagram is used during a download to determine which timeout values to use. The definition of each of the timeout types associated with the timeout commands is shown in Table 101.

Table 101 – Timeout command definitions

Command	Description
Initial Timeout	This command sets the download timeout used during the Initial State as shown in Figure 8. This timeout is measured from the time the overall package download began.
Initial Activity Timeout	This command sets an activity timeout to be used only during the Initial State as shown in Figure 8. The activity timeout is measured from the most recent time any package data had been transferred to the recipient. Note that during all states other than the Initial State, there is no activity timeout (the activity timeout is infinite).
Recoverable Timeout	This command sets the download timeout used during the Recoverable State as shown in Figure 8. This timeout is measured from the time the overall package download began.
Unrecoverable Timeout	This command sets the download timeout used during the Unrecoverable State as shown in Figure 8. This timeout is measured from the time the overall package download began.

E.4.8 Reboot Command

This command indicates that the recipient reboot in order to complete the installation process. If used, this command MUST be the last command in the command list (other than End, if present).

The Length parameter for this command MUST be 0 (zero), indicating that no Value field follows.

E.4.9 Format File System

This command indicates that the recipient reformat its file system as part of the installation process. If used, this command implies that all existing files in the file system (or the portion of the file system relevant for the installation process) are to be cleared and overwritten by the new files in the package.

The Length parameter for this command MUST be 0 (zero), indicating that no Value field follows.

E.4.10 Minimum and Maximum Version Commands

The Minimum Version and Maximum Version commands are used to specify the range of software version numbers for which the package is intended to apply.

When a minimum and/or maximum version number is specified in the package using these commands, the recipient MUST NOT install the files or take any other action specified in the command list if the software version of the recipient falls outside the indicated range.

This command MAY be used only if the format of the actual software version associated with the recipient is in a hierarchical format that can be compared numerically given the procedures outlined below.

The minimum and maximum version commands include information in the Value portion of the command. The format of this information is defined in Table 102.

Table 102 – Value format for the minimum and maximum version commands

Field	Type	Description
Version	Array of 32-bit integers	An array of integer elements indicating the version number. This is considered a hierarchical version number (e.g., “1.0.20.3”), where each successive integer represents a more minor element of the version number.

The following procedure is used to determine if a version is within the indicated range.

If a Minimum Version is given, then for each element of the Version array, beginning with the first (most major element):

1. If this element of the recipient’s actual version is greater than the corresponding element of the minimum version, then the recipient’s version meets the requirement and the procedure is complete.
2. If this element of the recipient’s actual version number is less than the corresponding element of the minimum version, then the recipient’s version does not meet the requirement. In this case, the procedure is complete and the recipient MUST NOT install the files in this package or follow any of the remaining commands.
3. Otherwise (the values are equal),
    a. If this is the last element in the array, then the recipient’s version meets the requirement and the procedure is complete.
    b. Otherwise (more elements remain), the procedure SHOULD continue at step 1 using the next element of the array.

If a Maximum Version is given, then for each element of the Version array, beginning with the first (most major element):

1. If this element of the recipient’s actual version is less than the corresponding element of the maximum version, then the recipient’s version meets the requirement and the procedure is complete.
2. If this element of the recipient’s actual version number is greater than the corresponding element of the maximum version, then the recipient’s version does not meet the requirement. In this case, the procedure is complete and the recipient MUST NOT install the files in this package or follow any of the remaining commands.
3. Otherwise (the values are equal),
    a. If this is the last element in the array, then the recipient’s version meets the requirement and the procedure is complete.
    b. Otherwise (more elements remain), the procedure SHOULD continue at step 1 using the next element of the array.

E.4.11 Role Command

The role command is used to indicate the target application or purpose of the package. This is intended to indicate any side effects or post-processing that might be required for a particular package.

The role commands include information in the Value portion of the command. The format of this information is defined in Table 103.

Table 103 – Value format for the role command

Field	Type	Description
Role	32-bit integer	An enumeration indicating the target application or purpose of the package. The following values are defined: 1 = Software upgrade 2 = Software recovery 3 = Web content 4 = Vendor configuration 5 = Tone file (see [28] Appendix B) 6 = Ringer file (see [28] Appendix B) Values with 0xFF as their most significant octet are to be interpreted as a vendor-specific Role. In this case, the subsequent three octets contain the OUI (organizationally unique identifier) identifying the vendor as defined in [13]. When this value is used, the vendor MAY define subsequent additional arguments to be included in this command in order to specifically identify the role. Any additional arguments are to be interpreted in a vendor-specific manner. All other values are reserved.

E.4.12 Minimum Storage Commands

The minimum storage commands include Minimum Volatile Storage Size and Minimum Non-Volatile Storage Size.

The minimum storage commands indicate the minimum requirement of the recipient device to be able to install the files contained in the package. If present, each command indicates the minimum requirement for the type of storage indicated by the command name.

If the recipient device does not meet a specified minimum requirement, the recipient MUST NOT install any of the files in the package or continue processing commands.

The minimum storage commands include information in the Value portion of the command. The format of this information is defined in Table 104.

Table 104 – Value format for the minimum storage commands

Field	Type	Description
Storage Size	32-bit Integer	The minimum required storage in bytes of the type indicated by the command.

E.4.13 Required Attributes Command

The Required Attributes command is used to specify additional attributes of the recipient device that are required in order for the package to be considered valid for installation.

One or more Required Attributes commands MAY be included in a single package, each indicating a different class of attributes required.

The Required Attribute command includes information in the Value portion of the command. The format of this information is defined in Table 105.

Table 105 – Value format for the required attributes command

Field	Type	Description
Defining Entity	32-bit Integer	Identifier indicating the definer of the Class and Attribute values used in this command. The following values are defined: A value of 0 (zero) indicates standard Class and Attribute definitions. Standard definitions are those defined by this version or future versions of this specification. Values with 0xFF as their most significant octet indicate vendor-specific Class and Attribute definitions. In this case, the subsequent three octets contain the OUI (organizationally unique identifier) identifying the vendor as defined in [13]. If a recipient processes a Required Attributes command with a defining entity value that it does not recognize, it SHOULD ignore the command and continue processing subsequent commands.
Class	32-bit Integer	An enumeration indicating the criterion for which the recipient is to be compared to determine whether or not this package is appropriate for that device. For a given criterion, the attribute array field indicates the particular allowed values associated with that criterion. In this version of the specification, no standard class values are defined. For vendor-specific defining entities, the interpretation of class values is vendor-specific. If a recipient processes a Required Attributes command with a class value that it does not recognize, it SHOULD ignore the command and continue processing subsequent commands.
Attribute Array	Array of 32-bit Integer	A variable-length array attribute, where each attribute is an enumeration of a particular allowed value for the particular class. If actual value associated with the recipient device matches any of the values listed in this array, then the recipient meets the specified requirement. Otherwise, the recipient does not meet the requirement and the package MUST NOT be installed. In this version of the specification, no standard attribute values are defined. For vendor-specific defining entities, the interpretation of attribute values is vendor-specific.

E.5 Signatures

The signature section immediately follows the command list section of the package file. The signature section consists of a digital signature block using the PKCS #7 signature syntax [19].

In particular, the signature block includes exactly one PKCS #7 SignedData Object, which contains zero or more signatures with the following constraints:

- The signatures are "external signatures," meaning that the signed message is not encapsulated within the SignedData Object. Instead, the signed message data consists of the octet string formed by the header and the command list components of the package.
- The contentType element of the contentInfo MUST indicate type "data."
- The content element of the contentInfo MUST be empty, since this is an external signature and the message data resides outside the signature itself.
- The digestAlgorithm used for each signature MUST be of type SHA-1.
- The digestEncryptionAlgorithm used for each signature MUST be of type RSA.

- The Tag value indicating the Identifier associated with the overall SignedData Object MUST be less than or equal to 30, resulting in a single-octet encoding of the Identifier.
- If there are no signatures in the signature block, there would be no extended certificates or certificate revocation lists, the SignerInfo set would be empty, and the digestAlgorithms set MAY be empty. All the other fields in SignedData MUST be present as normal. Note that the content of an empty signature block is independent of the content of the package and thus can be pre-computed as a fixed sequence of bytes.

If the signature block contains more than one signature, at least one of the signatures MUST be successfully validated for the recipient to consider the signed package as trusted.

If one or more signatures are expected by the package recipient, the recipient MUST validate the signature or signatures prior to processing the commands contained within the command list. If none of the included signatures are validated, the recipient MUST NOT process any of the commands in the command list or install any of the files contained in the package.

If the recipient implementation is such that command list validation and processing might be done without having loaded the entire package file from its source, the recipient MAY assume that the combined length of the header, command list, and signature block is no greater than 150 kilobytes.

Note that although the signed message data includes only the package header and command list, the signature assures the integrity of the entire package because all commands that refer to payload files include a hash of the file contents.

Note also that additional signatures can be added to an existing signed package file without modifying any part of the file other than the signature block itself. The package format is structured such that the other content (header, command list, and payload) of the package file need not change if the length of the signature block changes.

Annex F. Device-Gateway Association

F.1 Introduction

The CPE WAN Management Protocol can be used to remotely manage CPE Devices that are connected via a LAN through a Gateway. When an ACS manages both a Device and the Gateway through which the Device is connected, it can be useful for the ACS to be able to determine the identity of that particular Gateway.

The procedures defined in this Annex allow an ACS to determine the identity of the Gateway through which a given Device is connected.

As an example of when this capability might be needed, an ACS establishing QoS for a particular service might need to provision both the Device as well as the Gateway through which that Device is connected. To do the latter, the ACS would need to determine the identity of that particular Gateway.

The specific scenario that the defined mechanism is intended to accommodate is where both the Gateway and Device are managed via the CPE WAN Management Protocol, and both are managed by the same ACS (or by distinct ACSs that are appropriately coupled). Where a Device and Gateway are managed by independent ACSs, it is assumed that there is no requirement for either ACS to be made aware of the Device-Gateway association.

The defined mechanism relies on the Device's use of DHCP [23] / [38]. It is expected that the vast majority of remotely manageable Devices will use DHCP, though not necessarily all such Devices. While the mechanism defined here for Device-Gateway association requires the use of DHCP, a Device using this mechanism need not use DHCP for address allocation. This mechanism makes no assumptions about the address allocated to the Device. That is, the Device might have a private or public IP address.

F.1.1 Terminology

The following terminology is used in this Annex.

Device	CPE connected via local area network through a Gateway, bridge, or router.
Device Identity	A three-tuple that uniquely identifies a Device, which includes the manufacturer OUI, serial number, and (optionally) product class.
Gateway	Internet Gateway Device.
Gateway Identity	A three-tuple that uniquely identifies a Gateway, which includes the manufacturer OUI, serial number, and (optionally) product class.

F.2 Procedures

The procedures for Device-Gateway association are summarized as follows:

- A Device following this Annex will pass its Device Identity to the Gateway via a vendor-specific DHCP option. When the Gateway receives this information, it populates a table containing identity information for each Device on its LAN. This information is made available to the ACS via the ManageableDevice table in the Gateway's Data Model, defined in [27] and [35].
- In the DHCP responses, the Gateway provides the Device with its Gateway Identity, which the Device makes available to the ACS via the GatewayInfo data Object defined in [34] and [35]. The Device notifies the ACS of changes to the contents of this Object. Thus a Device connecting to a previously unknown Gateway will result in the ACS being notified of the Gateway Identity.
- To ensure the validity of this information, which is carried over an inherently insecure DHCP exchange, the ACS validates the Gateway Identity provided by the Device by crosschecking against the Device Identity provided by the Gateway.

F.2.1 Gateway Requirements

A Gateway conforming to this Annex MUST support the DeviceAssociation:2 profile as defined in [27] or the DeviceAssociation:1 profile as defined in [35], depending on the Root Data Model implemented.

A Gateway conforming to this Annex MUST inspect all DHCPv4 or DHCPv6 requests received on a LAN interface and determine if the requesting Device has included its Device Identity in the request. A DHCP request is determined to include the Device Identity if it contains a DHCPv4 V-I Vendor-Specific Information Option (option number 125, as defined in [25]) or DHCPv6 Vendor-Specific Information Option (option number 17, as defined in [38]) that includes the Device Identity information, as defined in Section F.2.5. The DHCPv4 requests for which this requirement applies are DHCPDISCOVER, DHCPREQUEST, and DHCPINFORM. The DHCPv6 requests for which this requirement applies are SOLICIT, REQUEST, RENEW, and INFORMATION-REQUEST.

If the DHCP request is determined to include the Device Identity, then the Gateway MUST do the following:

- The Gateway MUST include its Gateway Identity in all subsequent DHCP responses. The Gateway Identity is carried in the DHCPv4 V-I Vendor-Specific Information Option (option number 125, as defined in [25]) or DHCPv6 Vendor-Specific Information Option (option number 17, as defined in [38]), as defined in Section F.2.5. The DHCPv4 responses for which this requirement applies are DHCPOFFER and DHCPACK. The DHCPv6 responses for which this requirement applies are ADVERTISE and REPLY.
- On successful completion of the DHCP exchange, if an entry with a matching Device Identity is not currently listed in the ManageableDevice table, then the Gateway MUST add a new entry in its ManageableDevice table (see [27] and [35]) that includes the Device Identity for this Device.

The Gateway MUST adhere to the following additional requirements:

- The Gateway MUST retain a Device's entry in the ManageableDevice table as long as the Device remains actively connected to the Gateway's LAN.
- The Gateway MUST remove a Device's entry when either:
    - The DHCP-supplied information becomes invalid, e.g. the DHCPv4 lease expires or is released.
    - The Gateway determines that the Device is no longer actively connected to the Gateway's LAN using a locally defined means of connectivity detection.
- The Gateway MUST allow the ACS to request Active notification on additions or deletions to the ManageableDevice table. If the ACS has set the Notification Attribute for the Parameter ManagementServer.ManageableDeviceNumberOf-Entries to Active notification, then the Gateway MUST notify it each time a Device entry is added or removed using the Notification mechanism defined by the CPE WAN Management Protocol. If Active notification is enabled for this Parameter, the Gateway MUST limit the frequency of Active notification resulting from changes to the number of entries in the ManageableDevice table as specified by the value of the ManageableDeviceNotificationLimit Parameter in the same Object.

F.2.2 Device Requirements

A Device conforming to this Annex MUST support the GatewayInfo:1 profile as defined in [34] and [35].

A Device conforming to this Annex MUST do the following:

- In DHCP requests, the Device MUST include a DHCPv4 V-I Vendor-Specific Information Option (option number 125, as defined in [25]) or DHCPv6 Vendor-Specific Information Option (option number 17, as defined in [38]) that includes its Device Identity information, as defined in Section F.2.5. The DHCPv4 requests for which this requirement applies are DHCPDISCOVER, DHCPREQUEST, and DHCPINFORM. The DHCPv6 requests for which this requirement applies are SOLICIT, REQUEST, RENEW, and INFORMATION-REQUEST.
- If the DHCP response includes the Gateway Identity carried in the DHCPv4 V-I Vendor-Specific Information DHCP Option (option number 125, as defined in [25]) or DHCPv6 Vendor-Specific Information Option (option number 17, as defined in [38]), as defined in Section F.2.5, the Device MUST record the received value in the GatewayInfo data Object defined in [34] and [35]. All of the following values MUST be recorded:

  Device.GatewayInfo.ManufacturerOUI

  Device.GatewayInfo.SerialNumber

  Device.GatewayInfo.ProductClass

- The DHCPv4 responses for which this requirement applies are DHCPOFFER and DHCPACK. The DHCPv6 response for which this requirement applies are ADVERTISE and REPLY.
- If any of the elements of the Gateway Identity are not present in the V-I Vendor-Specific Information DHCP Option, the Device MUST record an empty string for each such item (replacing the previous value, if any).
- For all of the Parameters in the Device.GatewayInfo Object, the Device MUST by default set the Notification attribute as defined in Annex A to Active notification. The Device MUST apply this default whenever the URL of the ACS is set or subsequently modified. Whenever Active notification is enabled for these Parameters, the device MUST actively notify the ACS as defined in Annex A if the value of any of these Parameters changes.
- If the DHCP-discovered information becomes invalid, e.g. the DHCPv4 lease is released or expires without renewal, all entries in the GatewayInfo Object MUST be discarded (set to the empty string).

F.2.3 ACS Requirements

Whenever a Device is associated with a Gateway, the Device will notify the ACS, providing the new Gateway Identity information. When this occurs, the ACS SHOULD do the following:

- If the ACS has previously associated the Device with a Gateway, the ACS SHOULD examine the Gateway Identity from the Device (from the GatewayInfo Object) and compare it to the Gateway Identity of the prior association. If the association is unchanged, the ACS need not take any further action.
- If the Gateway Identity from the Device is different from the identity of the Gateway previously associated with the Device, or if there was no previous Gateway association for the Device, then the ACS SHOULD first validate the information provided by the Device, and if validated, update the Device-Gateway association to indicate the new Gateway Identity.

  The ACS SHOULD consider the association valid *only* if all elements of the Device Identity match the Device Identity elements in at least one entry in the ManageableDevice table of the indicated Gateway (see [27] and [35]). The ACS would determine the current contents of the ManageableDevice table either by contacting the Gateway using a Connection Request to read the table, or receiving Active notifications on additions and deletions to this table (by the ACS having previously requested Active notifications on the ManageableDeviceNumberOf-Entries Parameter).

F.2.4 Device-Gateway Association Flows

Note – The examples in this Section are specific to DHCPv4. The flows for DHCPv6 would display the same logic but with DHCPv4 messages replaced with the corresponding DHCPv6 messages.

Figure 9 shows the flow associated with the procedures for Device-Gateway association, where the Device uses a DHCP Discover message to initiate the association as part of DHCP address allocation.

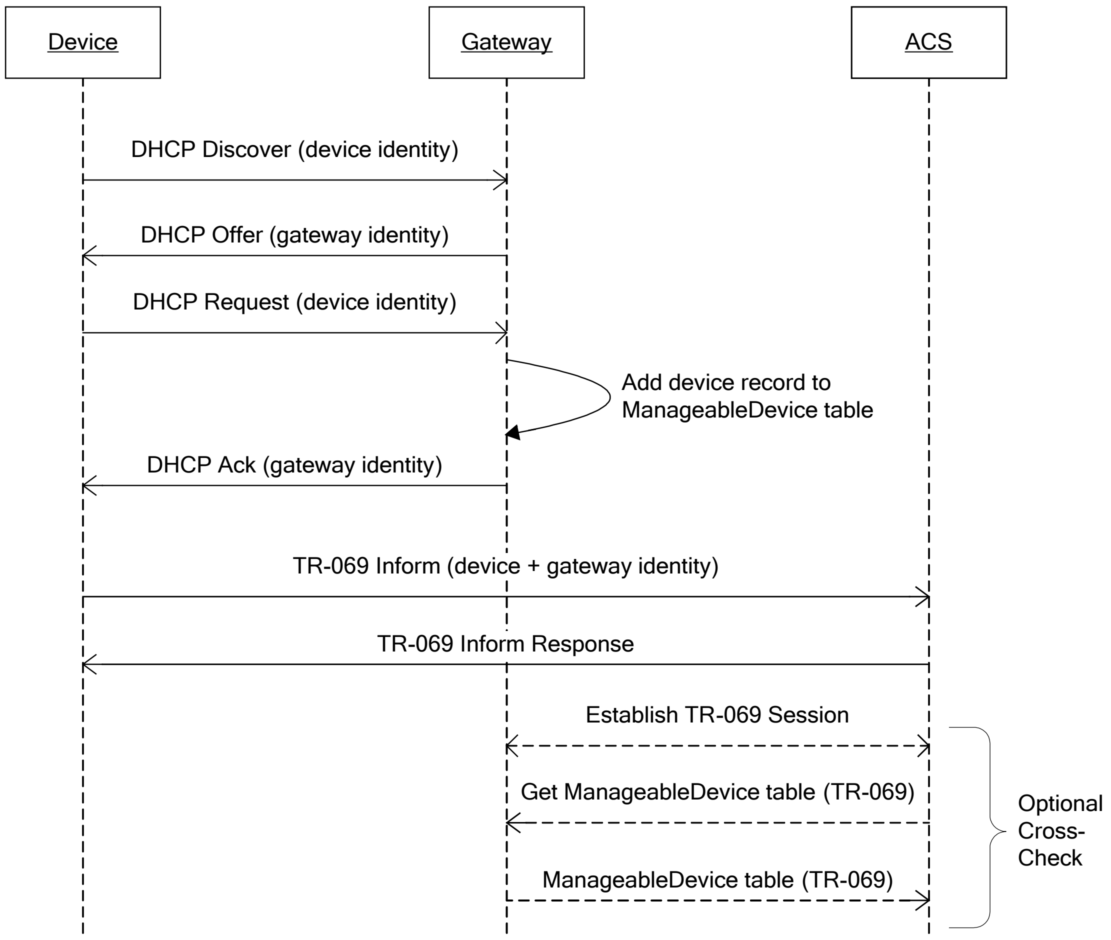

Device
Gateway
ACS
DHCP Discover (device identity)
DHCP Offer (gateway identity)
DHCP Request (device identity)
Add device record to ManageableDevice table
DHCP Ack (gateway identity)
TR-069 Inform (device + gateway identity)
TR-069 Inform Response
Establish TR-069 Session
Get ManageableDevice table (TR-069)
ManageableDevice table (TR-069)
Optional Cross-Check

Figure 9 – Device-Gateway Association using DHCP Discover

The use of DHCP does not dictate that the device use DHCP for address allocation. If the Device obtains IP addressing Parameters using other means, the device would use a DHCP Inform for the exchange of information with the Gateway. The flow for this case is show in Figure 10.

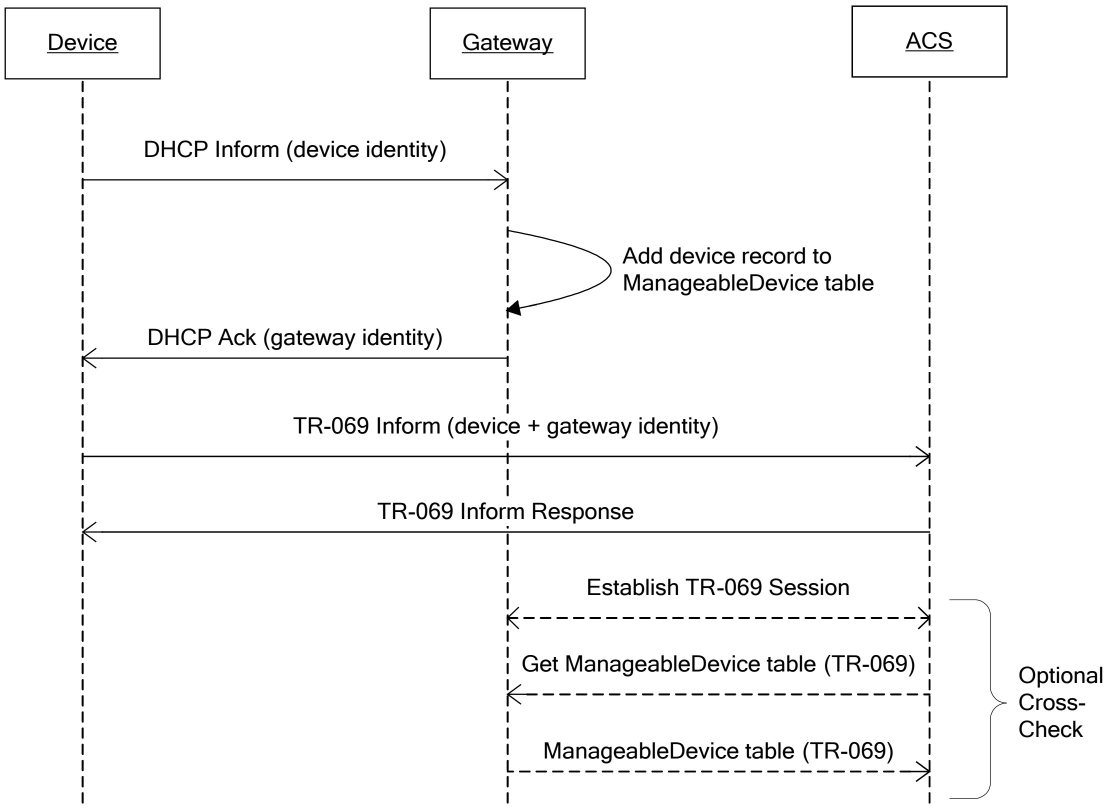

Device
Gateway
ACS
DHCP Inform (device identity)
Add device record to ManageableDevice table
DHCP Ack (gateway identity)
TR-069 Inform (device + gateway identity)
TR-069 Inform Response
Establish TR-069 Session
Get ManageableDevice table (TR-069)
ManageableDevice table (TR-069)
Optional Cross-Check

Figure 10 – Device-Gateway Association Using DHCP Inform

F.2.5 DHCP Vendor Options

The Device Identity and Gateway Identity information exchanged via DHCP MUST be contained within the DHCPv4 V-I Vendor-Specific Information Option (option number 125, as defined in [25]) or DHCPv6 Vendor-Specific Information Option (option number 17, as defined in [38]). These DHCP options are defined to allow vendor-specific information from multiple distinct organizations, where the specific organization is explicitly identified via an IANA Enterprise Number.

For DHCP messages that contain Device Identity or Gateway Identity information, the Vendor-Specific Information DHCP Option MUST include an element identified with the IANA Enterprise Number for the Broadband Forum that follows the format defined below. The IANA Enterprise Number for the Broadband Forum is **3561** in decimal (the "ADSL Forum" entry in the IANA Private Enterprise Numbers registry [21]).

Each vendor-specific element within this DHCP Option is defined to contain a series of one or more Encapsulated Vendor-Specific Option-Data fields, encoded as specified in [25] / [38]. Each such field includes a Sub-Option Code, a Sub-Option Length, and Sub-Option Data. The values for these elements defined in this Annex are listed in Table 106.

Table 106 – Encapsulated Vendor-Specific Option-Data fields

Encapsulated Option	Sub-Option Code		Source Entity	Source Parameter[27]
	DHCPv4 Option 125	DHCPv6 Option 17		
DeviceManufacturerOUI	1	11	Device	Device.DeviceInfo.ManufacturerOUI[28]
DeviceSerialNumber	2	12	Device	Device.DeviceInfo.SerialNumber[28]
DeviceProductClass	3	13	Device	Device.DeviceInfo.ProductClass[28]
GatewayManufacturerOUI	4	14	Gateway	DeviceInfo.ManufacturerOUI[29]
GatewaySerialNumber	5	15	Gateway	DeviceInfo.SerialNumber[29]
GatewayProductClass	6	16	Gateway	DeviceInfo.ProductClass[29]

Note – the DHCPv6 Option 17 Sub-Option Codes were changed in TR-069 Amendment 5 (previously they were 1-6) because they overlapped with the Section 3.1 ACS DHCP Discovery Codes (1-4).

In encoding the source Parameter value in the corresponding Sub-Option Data element, the resulting string MUST NOT be null terminated.

For a DHCP request from the Device that contains the Device Identity, the DHCP Option MUST contain the following Encapsulated Vendor-Specific Option-Data fields:

- DeviceManufacturerOUI
- DeviceSerialNumber
- DeviceProductClass (this MAY be left out if the corresponding source Parameter is not present)

For a DHCP response from the Gateway that contains the Gateway Identity, the DHCP Option MUST contain the following Encapsulated Vendor-Specific Option-Data fields:

- GatewayManufacturerOUI
- GatewaySerialNumber
- GatewayProductClass (this MAY be left out if the corresponding source Parameter is not present)

F.3 Security Considerations

While this Annex was designed to provide a high degree of security, some known vulnerabilities remain:

- While the mechanism to allow the ACS to validate the identity information provided to it by the Device is optional, it is strongly encouraged that this

[27] The value of the corresponding Sub-Option Data element is obtained from the specified Parameter value.
[28] As defined in [34] and [35].
[29] As defined in [27] and [35].

validation be implemented. The use of this validation is the only means within the context of this Annex to overcome the lack of an inherent integrity checking mechanism in the DHCP exchange between the Device and Gateway. By using this validation, attempts to tamper with the identity information of either the Device or Gateway can be detected by the ACS.

- The condition for validation of the Device-Gateway association is that the Device can communicate over the LAN to the Gateway and that the Device and Gateway can authenticate themselves via the CPE WAN Management Protocol to the ACS. The possibility exists that a valid Device not present on a Gateway's LAN could falsify its association with a Gateway by providing a communication path between the Device and the Gateway's LAN. For example, a Device could establish a communication path to a server, which in turn communicates with a Trojan horse application on the target LAN, which acts as a proxy for the Device. Providing such a path could make the Device indistinguishable from one physically connected to the LAN. To mitigate this possibility, the Gateway can optionally provide mechanisms to allow the user to monitor and regulate what devices are present on the LAN.

Annex G. Connection Request via NAT Gateway

Note – The mechanism defined in this Annex is OBSOLETED in favor of the "XMPP Connection Request" mechanism defined in Annex K. This mechanism only works with "Classic STUN" as defined in RFC 3489 [24], which has been made obsolete by the introduction of RFC 5389 [36]. This mechanism was not designed to work with STUN as defined in RFC 5389. IPv6 deployments will either not use NAT or will use it in different ways.

G.1 Introduction

The CPE WAN Management Protocol can be used to remotely manage CPE Devices that are connected via a LAN through a Gateway. When an ACS manages a Device connected via a NAT Gateway (where the Device has been allocated a private IP address), the CPE WAN Management Protocol can still be used for management of the Device, but with the limitation that the Connection Request mechanism defined in Section 3.2.2 that allows the ACS to initiate a Session cannot be used.

The procedures defined in this Annex allow an ACS to initiate a Session with a device that is operating behind a NAT Gateway. This provides the equivalent functionality of the Connection Request defined in Section 3.2.2, but makes use of a different mechanism to accommodate this scenario.

The mechanism defined in this Annex does *not* assume that the Gateway through which the Device is connected supports the CPE WAN Management Protocol. This mechanism requires support only in the Device and the associated ACS.

G.2 Procedures

To accommodate the ability for an ACS to issue the equivalent of a Connection Request to CPE allocated a private address through a NAT Gateway that might not be CPE WAN Management Protocol capable, the following is required:

- The CPE MUST be able to discover that its connection to the ACS is via a NAT Gateway that has allocated a private IP address to the CPE.
- The CPE MUST be able to maintain an open NAT binding through which the ACS can send unsolicited packets.
- The CPE MUST be able to determine the public IP address and port associated with the open NAT binding, and communicate this information to the ACS.

To accomplish the above items, this Annex defines a particular use of the STUN mechanism, defined in RFC 3489 [24].

The use of STUN for this purpose requires that a new UDP-based Connection Request mechanism be defined to augment the existing HTTP-based Connection Request mechanism defined in Section 3.2.2.

The procedures for making use of STUN to allow the use of UDP Connection Requests to a CPE are summarized as follows:

- The ACS enables the use of STUN in the CPE (if it is not already enabled by factory default) and designates the STUN server for the CPE to use.
- The CPE uses STUN to determine whether or not the CPE is behind a NAT Gateway with a private allocated address.
- If the CPE is behind a NAT Gateway with a private allocated address, the CPE uses the procedures defined in STUN to discover the binding timeout.
- The CPE sends periodic STUN Binding Requests at a sufficient frequency to keep alive the NAT binding on which it listens for UDP Connection Requests.
- When the CPE determines the public IP address and port for the NAT binding on which it is listening for UDP Connection Requests, and whenever it subsequently changes, the CPE communicates this information to the ACS. Two means are provided by which the ACS, at its discretion, can obtain this information—either from information provided in the STUN Binding Request messages themselves, or via Notification on changes to the UDPConnectionRequestAddress Parameter, which the CPE will update to include the public Connection Request address and port.
- Whenever the ACS wishes to establish a connection to the CPE, it can send a UDP Connection Request to the CPE. To accommodate the broadest class of NAT Gateways, this will be sent from the same source address and port as the STUN server.

G.2.1 CPE Requirements

A CPE conforming to this Annex MUST support the UDPConnReq :1 profile as defined in [27] and [35] if the CPE is an Internet Gateway Device, or as defined in [34] and [35] if the CPE is any other type of Device.

Whenever the STUNEnable Parameter in the ManagementServer Object is set to true, CPE following the requirements of this Annex MUST make use of the procedures defined in STUN [24] to determine whether or not address and/or port translation is taking place between the CPE and the STUN server. If address and/or port translation is taking place, the CPE MUST:

- Determine the public IP address and port for the NAT binding on which it is listening for UDP Connection Request messages.
- Discover the NAT binding timeout, and send STUN Binding Request messages at a rate necessary to keep alive this binding.
- Indicate via STUN optional attributes on which binding it is listening for UDP Connection Requests, and if the binding has recently changed. Also, update the

UDPConnectionRequestAddress Parameter to indicate the current public IP address and port associated with the binding.

- Listen for UDP Connection Request messages, and act on these messages when they arrive.

The details of each of these functions are defined in the following Sections.

Note – While the CPE requirements defined here certainly apply to a Device connected via LAN to a Gateway, the same procedures can be followed by a Gateway, which might be operating behind a network-based NAT gateway. Thus the requirements are defined generically for CPE, which might be either a Device or Gateway.

G.2.1.1 **Binding Discovery**

When STUN is enabled via the STUNEnable Parameter in the ManagementServer Object, the CPE MUST send Binding Request messages to the STUN server designated in the STUNServerAddress and STUNServerPort Parameters, as defined in [24]. If no STUNServerAddress is given, the address of the ACS determined from the host portion of the ACS URL MUST be used as the STUN server address.

For the purpose of binding discovery, Binding Requests MUST be sent from the source address and port on which the CPE will be listening for UDP Connection Requests if it determines that address and/or port translation is in use (Binding Requests for binding timeout discovery, will be sent from a different port as described in Section G.2.1.2).

The basic Binding Request message allows the CPE to determine if address and/or port translation is in use between the CPE and the STUN server. This is determined by comparing the source address and port on which the request was sent to the MAPPED-ADDRESS attribute received in a response from the STUN server. If either the address or port is different, then translation is in use.

If it is determined that address and/or port translation is in use, the CPE MUST record the value of the MAPPED-ADDRESS attribute in the most recently received Binding Response. This represents the public IP address and port to which UDP Connection Requests would be sent.

Each time the CPE subsequently sends a Binding Request for the purpose of maintaining the binding (see G.2.1.2), the CPE MUST again determine if address and/or port translation is in use, and if so, obtain the public IP address and port information from the MAPPED-ADDRESS attribute in a successful Binding Response. The actions the CPE will take when this information changes are defined in Section G.2.1.3.

If the CPE has been provisioned with a STUNUsername and STUNPassword in the ManagementServer Object, then if the CPE receives a Binding Error Response from the STUN server with a fault code of 401 (Unauthorized), then the CPE MUST resend the Binding Request with the USERNAME and MESSAGE-INTEGRITY attributes as defined in [24]. Whenever a Binding Request is sent that includes the MESSAGE-INTEGRITY attribute, the CPE MUST discard a corresponding Binding Response if the MESSAGE-INTEGRITY attribute in the Binding Response is either invalid, as defined in [24], or is not present.

If the local IP address allocated to the CPE changes, the CPE MUST re-discover the binding using the procedures described above. The minimum limit on the Binding Request period defined by STUNMinimumKeepAlivePeriod does *not* apply in this case.

Other than Binding Request messages sent explicitly in response to a Binding Error Response from the STUN server with a fault code of 401 (Unauthorized), the CPE MUST NOT include the MESSAGE-INTEGRITY attributes in any Binding Request.[30]

The STUN client in the CPE need not support the CHANGE-REQUEST attribute of STUN Binding Requests, nor need it understand the CHANGED-ADDRESS, SOURCE-ADDRESS, and REFLECTED-FROM attributes present in a Binding Response.[31]

The STUN client in the CPE need not support the STUN messages for exchanging a Shared Secret. None of these messages are used in the application defined in this Annex.

G.2.1.2 Maintaining the Binding

To keep alive the NAT binding, the CPE MUST periodically retransmit Binding Request messages from the source address and port on which the CPE will be listening for UDP Connection Requests.

The CPE MUST NOT send these Binding Requests more frequently than is specified by the STUNMinimumKeepAlivePeriod Parameter in the ManagementServer Object.

The CPE MUST send these Binding Requests at least as frequently as is specified by the STUNMaximumKeepAlivePeriod Parameter in the ManagementServer Object, if a value is specified.

If the value of STUNMinimumKeepAlivePeriod and STUNMaximumKeepAlivePeriod are not equal, then the CPE MUST actively discover the longest keep-alive period for which the NAT binding is maintained. To do this, the CPE MUST use the procedures described generally in [24] to learn the binding timeout. Specifically, the CPE MUST be able to test whether the binding has timed out by sending Binding Requests from a secondary source port distinct from the primary source port, and use the RESPONSE-ADDRESS attribute in the Binding Request to indicate that the STUN Binding Response be sent to the primary source port (the port on which the CPE is listening for UDP Connection Request messages).

The specific procedures by which the CPE uses Binding Requests from the secondary source port to determine the binding timeout is left to the discretion of the CPE vendor. In general, the procedure would consist of two phases: a discovery phase, and a monitoring phase. During the discovery phase, the CPE is attempting to learn the value of the binding timeout, and would test different timeout values to determine the actual timeout value (for example, using a binary search). During the monitoring phase, the CPE would periodically test the binding prior to refreshing it to determine if the binding

[30] Because the STUN specification requires the STUN server to use message integrity in its response if message integrity was used in the request, the CPE cannot use message integrity for Binding Requests on its own, but only when so directed by the STUN server. This is to ensure that the server has total discretion as to when and whether message integrity is to be used.

[31] These attributes are primarily intended to allow discovery of the type of NAT in use, which is not required for this Annex.

is still in place. If not, the CPE could then revert to the discovery phase to determine a new value for the binding.

The minimum limit on the Binding Request period defined by STUNMinimumKeep-AlivePeriod does *not* apply to Binding Requests sent from a secondary source port.

G.2.1.3 Communication of the Binding Information to the ACS

Two means are defined by which the ACS can be informed of the binding information. The CPE MUST support both methods.[32] The first method involves the use of optional STUN attributes sent in the Binding Requests. The second method involves the CPE updating the value of the UDPConnectionRequestAddress Parameter as the binding information changes.

Table 107 specifies a set of STUN attributes are defined for this application. These use Attribute Type values that are greater than 0x7FFF, which the STUN specification defines as "optional." STUN servers that do not understand optional attributes, are required to ignore them.

Table 107 – Optional STUN attributes used in Binding Request messages

Attribute Type	Name	Description

0xC001	CONNECTION-REQUEST-BINDING	Indicates the binding on which the CPE is listening for UDP Connection Requests.
		The content of the Value element of this attribute MUST be the following byte string:
		`0x64 0x73 0x6C 0x66`
		`0x6F 0x72 0x75 0x6D`
		`0x2E 0x6F 0x72 0x67`
		`0x2F 0x54 0x52 0x2D`
		`0x31 0x31 0x31 0x20`
		This corresponds to the following text string:[33]
		"dslforum.org/TR-111 "
		A space character is the last character of this string so that its length is a multiple of four characters.
		The Length element of this attribute MUST equal:
		`0x0014` (20 decimal)

0xC002	BINDING-CHANGE	Indicates that the binding has changed. This attribute contains no value. Its Length element MUST be equal to zero. This attribute MUST only be used where the CONNECTION-REQUEST-BINDING is also included.

A CPE MUST include the CONNECTION-REQUEST-BINDING attribute in every Binding Request message whose source address and port are the address and port on which it is listening for UDP Connection Request messages. In all other Binding Request messages, the CPE MUST NOT include this attribute.

[32] Defining two methods allows flexibility by the ACS in making the tradeoffs between these two approaches. Specifically, the STUN-based approach may require a tighter coupling between the ACS itself and the associated STUN server, while the Notification-based approach may result in greater communication overhead.

[33] This text string is used to allow an observer, including the NAT Gateway itself, to identify that these STUN messages represent UDP Connection Request bindings associated with this specification. A Gateway might use this knowledge to optimize the associated performance. For example, a Gateway could lengthen the UDP timeout associated with this binding to reduce the frequency of binding updates.

In every Binding Request message sent in which the CPE includes the CONNECTION-REQUEST-BINDING attribute, if the value of the STUNUsername Parameter in the ManagementServer Object is non-empty, the CPE MUST include the USERNAME attribute set to the value of the STUNUsername Parameter, if necessary padded with trailing spaces to make its length a multiple of 4 bytes (as required by the STUN protocol).

Whenever the CPE detects a change to the NAT binding (as well as the first time the CPE determines the binding), it MUST immediately send a Binding Request message from the primary source port (the port on which the CPE is listening for UDP Connection Request messages) that includes the BINDING-CHANGE attribute. This Binding Request MUST NOT include the RESPONSE-ADDRESS or CHANGE-REQUEST attributes. In all other Binding Request messages, the CPE MUST NOT include the BINDING-CHANGE attribute. The minimum limit on Binding Request period defined by STUNMinimum-KeepAlivePeriod does *not* apply to Binding Requests that include the BINDING-CHANGE attribute.

For Binding Requests that include the BINDING-CHANGE attribute, the CPE MUST follow the retransmission procedures define in [24] to attempt to ensure the successful reception. If, following these retransmission procedures, the CPE determines that the Binding Request has failed, it MUST NOT make further attempts to send Binding Requests that include the BINDING-CHANGE attribute (until the binding subsequently changes again).

When the CPE determines that address and/or port mapping is in use, and whenever the CPE determines that the binding has changed (as well as the first time the CPE determines the binding), the CPE MUST update the value of the UDPConnectionRequestAddress Parameter in the ManagementServer Object. Specifically:

- The Host portion of the UDPConnectionRequestAddress MUST be set to the current public IP address for the binding associated with the UDP Connection Request as determined from the most recent binding information.
- The Port portion of the UDPConnectionRequestAddress MUST be set to the current public port for the binding associated with the UDP Connection Request as determined from the most recent binding information.

When the CPE determines that address and/or port mapping is in use, the CPE MUST also set the NATDetected Parameter in the ManagementServer Object to true.

If the ACS has set the Notification attribute on the UDPConnectionRequestAddress Parameter to Active notification, then whenever the binding information has changed, the CPE MUST establish a connection to the ACS and include the UDPConnectionRequestAddress in the Inform message, as defined in Annex A.

When the UDPConnectionRequestAddress is changed, if the time since the most recent Notification on a change to the UDPConnectionRequestAddress is less than the value of UDPConnectionRequestAddressNotificationLimit, the Notification MUST be delayed until the specified minimum time period is met.

Note – In addition to the specified minimum notification period, the CPE MAY use its discretion to delay notifying the ACS of updated binding information in order to avoid excessive notifications. Such a delay would only be used if the CPE is confident that the binding is likely to change again within a brief period. For example, during active discovery of the binding timeout it is reasonable to expect frequent binding changes. Similarly, a CPE might be able to detect that a security attack is causing frequent binding changes, and limit the number of notifications until the attack ceases.

If the CPE determines that neither address nor port mapping are in use, then the CPE MUST indicate this to the ACS by setting the NATDetected Parameter to false, and setting the UDPConnectionRequestAddress such that the Host and Port are the local IP address and port on which the CPE is listening for UDP Connection Request messages.

G.2.1.4 UDP Connection Requests

A CPE conforming to this Annex MUST listen for UDP Connection Request messages on the port that it has designated for this purpose. This MUST be true whether or not the CPE has detected address or port translation in use, and whether or not the use of STUN is enabled.

Note – a CPE MUST also continue to listen for HTTP-based Connection Requests as defined in Section 3.2.2.

The format of the UDP Connection Request message is defined in Section G.2.2.3. When the CPE receives a UDP Connection Request message, it MUST both authenticate and validate the message.

A UDP Connection Request message is valid if and only if the following requirements are met:

- It MUST NOT violate any requirements specified for an HTTP 1.1 request message.
- The Method given in the Request Line MUST be “GET”.
- The Timestamp given by the value of the “ts” query string argument MUST be strictly greater than the Timestamp value for the UDP Connection Request message that had been most recently received, validated, and authenticated.

  To allow the above comparison to be made, the CPE MUST maintain a persistent record of Timestamp value of the most recent UDP Connection Request that was successfully validated and authenticated (except across CPE reboots). The Timestamp value for any UDP Connection Request message that fails to be validated or authenticated MUST NOT be recorded. The CPE MAY maintain a record of this most recent Timestamp across CPE reboots. If the CPE does not maintain this value across reboots, then immediately following the reboot the value zero MUST be used.

  The CPE MAY place stricter requirements on the Timestamp than stated above. The CPE MAY, for example, additionally verify that the Timestamp is within a time window relative to its understanding of the current time. If a CPE chooses to do this, it SHOULD avoid making the time window too narrow, in order to allow for a reasonable margin of error in both the CPE and ACS.

- The Message ID given by the value of the “id” query string argument MUST be distinct from that of the UDP Connection Request message that had been most recently received, validated, and authenticated.
- The Username given by the value of the “un” query string argument MUST match the value of the Parameter ManagementServer.ConnectionRequest-Username.

A UDP Connection Request message is authenticated if and only if the following requirements are met:

- The Signature given by the value of the “sig” query string argument MUST match the value of the signature locally computed by the CPE following the procedure specified in Section G.2.2.3 using the local value of the Parameter ManagementServer.ConnectionRequestPassword.

Whenever a CPE receives and successfully authenticates and validates a UDP Connection Request, it MUST follow the same requirements as for a HTTP-based Connection Request that are defined in Section 3.2.2.

The CPE MUST ignore a UDP Connection Request that is not successfully authenticated or validated.

The CPE MUST ignore the content of any non-empty Message Body that might be present in the UDP Connection Request (this allows the possibility of the use of a non-empty message body in a future version of this protocol).

Because STUN responses and UDP Connection Requests will be received on the same UDP port, the CPE MUST appropriately distinguish STUN messages from UDP Connection Requests using the content of the messages themselves. As the first byte of all STUN messages defined in [24] is either 0 or 1, and the first byte of the UDP Connection Request is always an ASCII encoded alphabetic letter, the CPE MAY use this distinction to distinguish between these messages.

Port 7547 has been assigned by IANA for the CPE WAN Management Protocol (see [20]), and the CPE MAY use this port for UDP Connection Requests.

G.2.2 ACS Requirements

An ACS following the requirements of this Annex MUST be associated with a STUN server that follows the requirements defined in this Section.

G.2.2.1 STUN Server Requirements

The STUN server MUST conform to all of the requirements defined in [24], with the following exceptions, which the STUN server MAY choose not to implement.

- The STUN server need not support the Shared Secret exchange mechanism defined in [24]. If message integrity is used, the shared secrets MUST be statically provisioned, and correspond to the STUNUsername and STUNPassword Parameters in the ManagementServer Object in the CPE.
- The STUN server need not support a secondary source IP address or port for sending Binding Responses (A2/P2). If it does not, the CHANGED-ADDRESS

attribute SHOULD be filled in with the primary address and port (A1/P1), and the STUN server MAY ignore the CHANGE-REQUEST attribute if received in a Binding Request.

The STUN server MAY require message integrity for any received Binding Requests of its choosing by responding to the request with a Binding Error Response with fault code 401 (Unauthorized).

G.2.2.2 Determination of the Binding Information

The ACS can choose either of the two defined mechanisms to determine the current binding information from a CPE.

G.2.2.2.1 STUN-based Approach

If the ACS chooses to use the attributes received by the STUN server, it SHOULD set a non-empty STUNUsername and STUNPassword in the ManagementServer Object of each CPE. The STUNUsername MUST be unique among all CPE managed by the corresponding ACS to ensure that the CPE can be distinguished. The STUNPassword SHOULD be unique among all CPE managed by the corresponding ACS, and SHOULD follow the password strength guidelines specified in [24].

Whenever the STUN server receives a Binding Request that includes both the BINDING-CHANGE and CONNECTION-REQUEST-BINDING attributes:

- The STUN server SHOULD respond with a Binding Error Response with fault code 401 (Unauthorized) in order to force the CPE to retransmit the Binding Request with message integrity included.
- When the STUN server receives the retransmitted request with message integrity, it SHOULD authenticate the requester. This would likely involve communication between the STUN server and ACS if they were not implemented as a single entity.
- If the authentication fails, the STUN server MUST respond with a Binding Request Error as defined in [24] and take no further action.
- If the authentication is successful, the STUN server SHOULD extract the source IP address and port from the Binding Request message, and record these as the new IP address and port to be used for UDP Connection Requests. Depending on the implementation, this might involve the STUN server informing the ACS of the IP address and port along with the corresponding STUNUsername, from which the ACS would then record this information for the CPE corresponding to that STUNUsername.
- The STUN server SHOULD perform the above only once for a given Transaction ID in the Binding Request. Redundant copies of the Binding Request with the same Transaction ID SHOULD be ignored.

Using this approach, the STUN server MAY choose not to require message integrity or authenticate any Binding Requests other than those for which it follows the above procedures to determine the binding information.

The ACS MAY determine the current binding at any time even if no change was notified by following the above procedure on any received Binding Request for which the CONNECTION-REQUEST-BINDING attribute is present. The required presence of the USERNAME attribute in these Binding Requests allows the ACS to tentatively determine the CPE's identity prior to subsequent authentication. This allows an ACS to periodically verify the binding information to ensure that it is up-to-date in case explicit indications of a binding change had failed to reach the ACS.

If the ACS determines that the CPE is no longer behind a NAT that is doing address or port mapping, the ACS MAY use HTTP-based Connection Requests as defined in Section 3.2.2.

G.2.2.2.2 *Notification-based Approach*

If the ACS chooses to use Active notification on the UDPConnectionRequestAddress Parameter, it SHOULD do the following:

- Set the Notification attribute for the UDPConnectionRequestAddress Parameter to Active notification.
- Record changes to the UDPConnectionRequestAddress Parameter whenever this Parameter is included in the Inform message, and use the most recently recorded value to determine the destination of UDP Connection Request messages. Specifically, the destination IP address for UDP Connection Request messages is determined from the "host" portion of this Parameter, and the destination port is determined from the "port" portion of this Parameter. If the host is given as a domain name, the ACS MUST use DNS to determine the associated IP address. If the port is not explicitly given in the UDPConnectionRequestAddress Parameter, port 80 MUST be used as the default value.
- Observe the value of the NATDetected Parameter (either by reading it when UDPConnectionRequestAddress changes, or by enabling Active notification on this Parameter as well). Whenever this Parameter is false, the ACS MAY use HTTP-based Connection Requests as defined in Section 3.2.2.

Using this approach, the ACS MAY choose not to require message integrity or authenticate any STUN Binding Requests, since these requests are not used to convey information to the ACS. In this case, the ACS need not set a STUNUsername or STUNPassword in the CPE.

G.2.2.3 **UDP Connection Requests**

The ACS MUST send UDP Connection Request messages from the same source IP address and port as the STUN server.

A UDP Connection Request message MUST be transmitted within a single UDP packet sent to the IP address and port determined by the ACS as described in Section G.2.2.2.

The ACS SHOULD send multiple copies of the same UDP Connection Request message in order to reduce the likelihood that the message is lost due to packet loss. When an ACS sends multiple copies of the same UDP Connection Request, the content of the message (including the message ID, timestamp, and cnonce, as defined below) MUST be identical for each successive copy.

There is no response message associated with a UDP Connection Request message.

The format of the UDP Connection Request message is derived from the format of an HTTP 1.1 GET message, though the HTTP 1.1 protocol itself is not used. Specifically, the UDP Connection Request message MUST conform to the following requirements:

- It MUST be a valid HTTP 1.1 GET message.
- It MUST contain no Message Body.
- If a Content-Length header is present, its value MUST be zero.
- The Method given in the Request Line MUST be "GET".
- The Request-URI given in the Request Line MUST be an Absolute-URI according to the rules defined in [15]. The URI MUST be formed as follows:
    - The Scheme portion of the URI MUST be "http" or "HTTP".
    - The Authority portion of the URI MUST be as specified in [15]. The ACS MAY set this to the value of ManagementServer.UDPConnectionRequest-Address, if it is known. Otherwise, the ACS MUST derive this string from the actual destination IP address and port to which the UDP Connection Request message will be sent. The "port" portion of this string MUST be present unless the destination port number is "80".
    - The Path portion of the URI MUST be empty.
    - The Query portion of the URI MUST contain a query string encoded as defined by the "application/x-www-form-urlencoded" content type defined in [26]. The query string MUST contain the following name-value pairs:

Name	Value
ts	Timestamp. The number of seconds since the Unix epoch until the time the message is created (the standard Unix timestamp).
id	Message ID. An unsigned integer value that MUST be set to the same value for all retransmitted copies of the same UDP Connection Request. The value MUST change between successive distinct UDP Connection Requests.
un	Username. The value of the Parameter ManagementServer.ConnectionRequest-Username as read from the CPE.
cn	Cnonce. A random string chosen by the ACS.
sig	Signature. Formed from the 40-character hexadecimal representation (case insensitive) of HMAC-SHA1 (Key, Text) [22], where: • Key is the value of the Parameter ManagementServer.ConnectionRequest-Password as read from the CPE. • Text is a string formed by concatenating the following elements (in the order listed, with no spaces between items): • The value of the ts (Timestamp) element • The value of the id (Message ID) element • The value of the un (Username) element • The value of the cn (Cnonce) element

Below is an example Request-URI:

```
http://10.1.1.1:8080?ts=1120673700&id=1234&un=CPE57689
&cn=XTGRWIPC6D3IPXS3&sig=3545F7B5820D76A3DF45A3A509DA8D8C38F13512
```

G.2.3 Message Flows

The following figures show example message flows associated with the procedures defined in Sections G.2.1 and G.2.2 to support Connection Requests to devices behind a NAT gateway.

In all of the examples, the address/port pairs use the notation *(A, P)*, where *A* is the IP address and *P* is the port. In the examples, the CPE uses *(A1, P1)* as its primary port (the port on which the CPE is listening for UDP Connection Request messages) and *(A1, P2)* is its secondary port (used for binding timeout discovery). When passing through a NAT Gateway, these addresses are translated to *(A1', P1')* and *(A1', P2')*, respectively. In all of the examples it is assumed that the STUN Server does not have a secondary address/port and thus the CHANGED-ADDRESS attribute in the Binding Response (which need not be used by the CPE) contains its primary address/port, *(A3, P3)*.

Figure 11 shows the periodic binding discovery and binding maintenance flows where the CPE sends the Binding Request from the primary source port and includes the CONNECTION-REQUEST-BINDING and (if a Username had been set) USERNAME attributes. In this example it is assumed that the STUN Server has not chosen to authenticate the request.

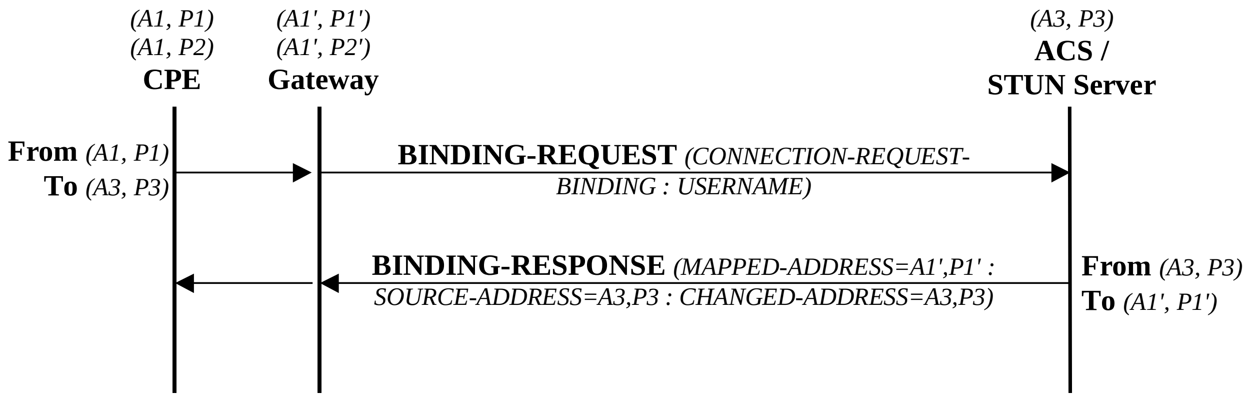

(A1, P1)
(A1, P2)
CPE
(A1', P1')
(A1', P2')
Gateway
(A3, P3)
ACS /
STUN Server
From (A1, P1)
To (A3, P3)
BINDING-REQUEST (CONNECTION-REQUEST-BINDING : USERNAME)
BINDING-RESPONSE (MAPPED-ADDRESS=A1',P1' : SOURCE-ADDRESS=A3,P3 : CHANGED-ADDRESS=A3,P3)
From (A3, P3)
To (A1', P1')

Figure 11 – Binding discovery / maintenance from the primary source port

Figure 12 shows a Binding Request sent by the CPE from its secondary source port for the purpose of discovering whether or not the primary binding has timed out in the NAT gateway. In this case the Binding Request does not include the CONNECTION-REQUEST-BINDING attribute since it is not sent from the primary source port. The last leg of the exchange (shown in grey) will not occur if the primary binding has timed out.

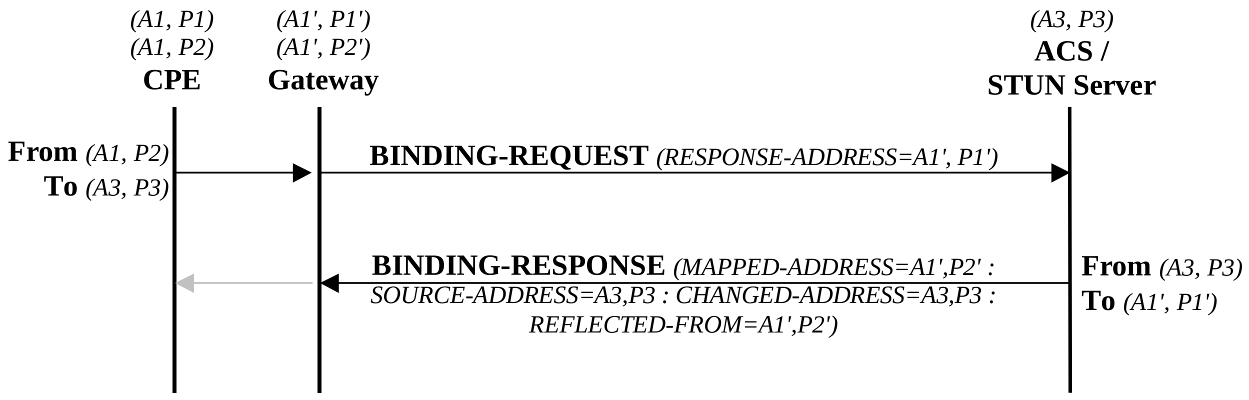

(A1, P1)
(A1, P2)
CPE
(A1', P1')
(A1', P2')
Gateway
(A3, P3)
ACS /
STUN Server
From (A1, P2)
To (A3, P3)
BINDING-REQUEST (RESPONSE-ADDRESS=A1', P1')
BINDING-RESPONSE (MAPPED-ADDRESS=A1',P2' : SOURCE-ADDRESS=A3,P3 : CHANGED-ADDRESS=A3,P3 : REFLECTED-FROM=A1',P2')
From (A3, P3)
To (A1', P1')

Figure 12 – Binding Request from secondary source port for binding timeout discovery

Figure 13 shows a Binding Change notification where the STUN Server has chosen to make use of the STUN-based approach (see Section G.2.2.2.1), and therefore authenticates the Binding Request prior to storing the information associating the Username with the current binding address and port.

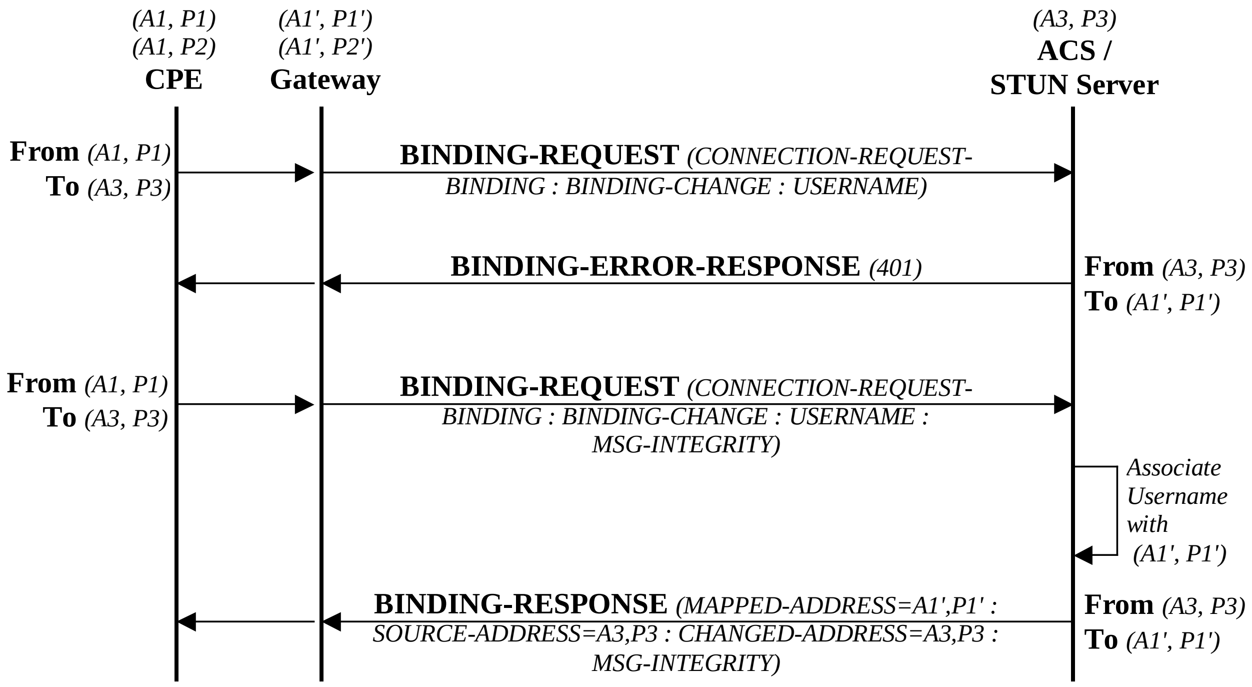

(A1, P1)
(A1, P2)
CPE
(A1', P1')
(A1', P2')
Gateway
(A3, P3)
ACS /
STUN Server
From (A1, P1)
To (A3, P3)
BINDING-REQUEST (CONNECTION-REQUEST-BINDING : BINDING-CHANGE : USERNAME)
BINDING-ERROR-RESPONSE (401)
From (A3, P3)
To (A1', P1')
From (A1, P1)
To (A3, P3)
BINDING-REQUEST (CONNECTION-REQUEST-BINDING : BINDING-CHANGE : USERNAME : MSG-INTEGRITY)
Associate Username with (A1', P1')
BINDING-RESPONSE (MAPPED-ADDRESS=A1',P1' : SOURCE-ADDRESS=A3,P3 : CHANGED-ADDRESS=A3,P3 : MSG-INTEGRITY)
From (A3, P3)
To (A1', P1')

Figure 13 – Binding change notification authenticated by the ACS

Figure 14 shows a Binding Change notification where the STUN Server has chosen to make use of the Notification-based approach (see Section G.2.2.2.2), and therefore does not need to authenticate the Binding Request since the ACS instead uses CPE WAN Management Protocol Notification to update the binding information.

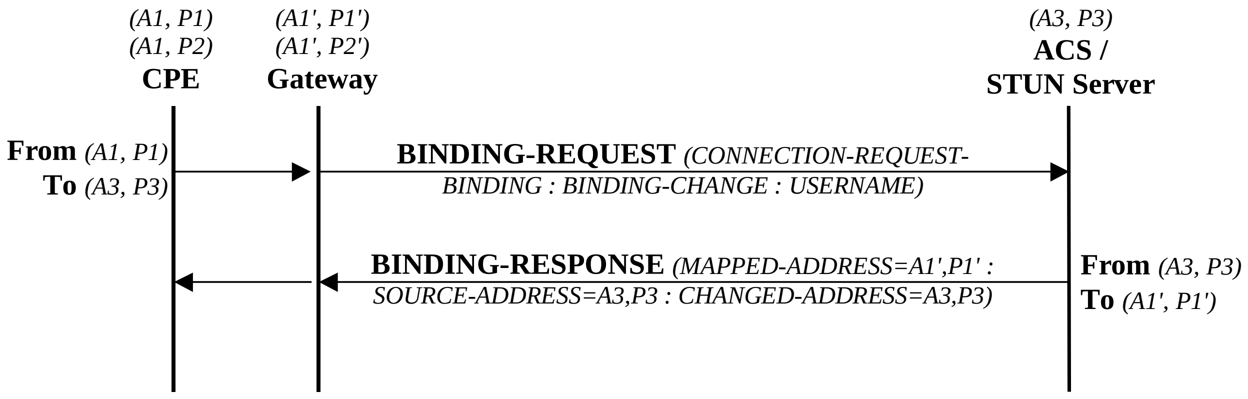

(A1, P1)
(A1, P2)
CPE
(A1', P1')
(A1', P2')
Gateway
(A3, P3)
ACS /
STUN Server
From (A1, P1)
To (A3, P3)
BINDING-REQUEST (CONNECTION-REQUEST-BINDING : BINDING-CHANGE : USERNAME)
BINDING-RESPONSE (MAPPED-ADDRESS=A1',P1' : SOURCE-ADDRESS=A3,P3 : CHANGED-ADDRESS=A3,P3)
From (A3, P3)
To (A1', P1')

Figure 14 – Binding change notification *not* authenticated by the ACS

Figure 15 shows a UDP Connection Request message sent to the CPE to initiate a CPE WAN Management Protocol Session. In this example, the STUN Server sends the identical UDP Connection Request multiple times to improve the likelihood of successful reception by the CPE.

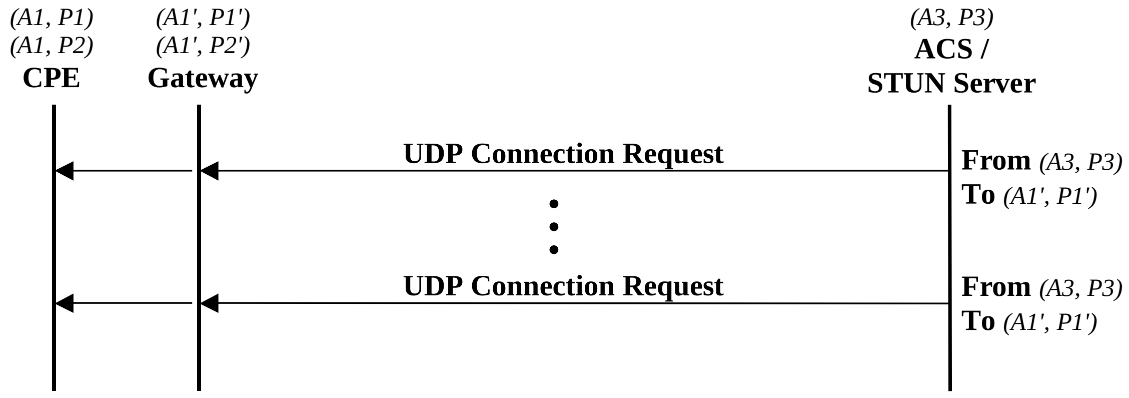


Figure 15 – UDP Connection Request

G.3 Security Considerations

The following security considerations associated with the procedures defined in this Annex are identified:

- The STUN specification describes several potential attacks using the STUN mechanism. The reader is referred to Section 12 of RFC 3489 [24] for a detailed description of these potential attacks and the associated risk.
- Because binding changes will result in actions required by the ACS—authentication of a CPE, and subsequent database update, and potentially establishment of a CPE WAN Management Protocol Session over which to receive an Inform—attacks that can cause frequent changes to the NAT binding could result in an increased burden on the ACS. The ACS can set a minimum limit on the rate of Notifications on binding changes if Active notification is used. However, there is a tradeoff between the maximum Notification rate and the length of time for which the ACS might not be able to send Connection Requests to the CPE due to out-of-date information.

Annex H.

Note – The Software Module Management UUID Usage originally defined in this annex, was moved to TR-181 Issue 2, Device Data Model for TR-069, Broadband Forum Technical Report [35].

Annex I.

Annex I is intentionally left blank

Annex J. CWMP Proxy Management

J.1 Introduction

CWMP can be extended to devices that do not have a native CWMP Endpoint of their own, but instead support another management protocol or “Proxy Protocol”. A CPE Proxier is a CPE that supports a CWMP Endpoint(s) and also supports one or more Proxy Protocols (example services include UPnP DM, Z-Wave etc.). A CPE Proxier uses these Proxy Protocols to manage the devices connected to it, i.e. the Proxied Devices. This approach is designed to support Proxy Protocols of all types that can exist in the CPE network now or in the future.

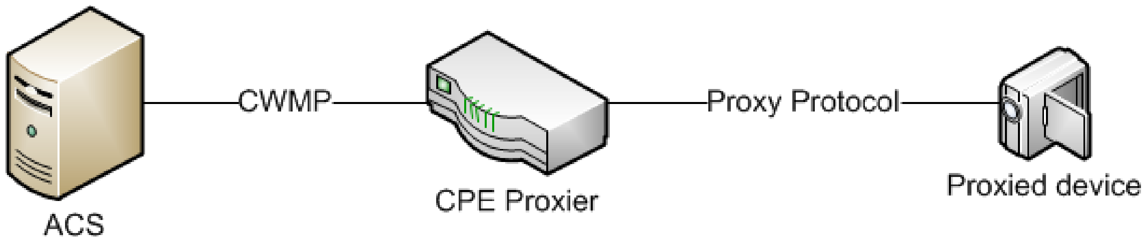


Figure 16 – Proxy management terminology

The function of the CPE Proxier is to seamlessly incorporate all of the elements and mechanisms/methods of the Proxy Protocol(s). Independent of the implementation, the ACS manages the Proxied Device through CWMP mechanisms, and is not aware of any Proxy Protocol commands that may be utilized to complete the requested actions.

In order to support a wide range of Proxy Protocols and devices, CWMP has two ways to model a Proxied Device; as a Virtual CWMP Device and using an Embedded Object.

A Virtual CWMP Device is used to model a more complex Proxied Device such as a bridge, router, Set Top Box or devices of similar type. The Virtual CWMP Device Mechanism represents a Proxied Device with a CWMP Endpoint in the CPE Proxier.

An Embedded Object is used to model a simpler Proxied Device such as a binary sensor, power switch or devices of similar type. The Embedded Object Mechanism utilizes an Embedded Object or Service Object within the CPE Proxier itself to represent the Proxied Device.

This Annex describes each CPE Proxier Mechanism and gives guidelines for dictating which approach might be appropriate for a given Proxy Protocol.

J.2 The Virtual CWMP Device Mechanism

The Virtual CWMP Device Mechanism provides the ACS with a CWMP Endpoint terminated on the CPE Proxier for the Proxied Device. The Virtual CWMP client will function with the same requirements as a CWMP client. The CPE Proxier that is supporting a Proxied Device with a Virtual CWMP Device is responsible for creating, supporting and sustaining a CWMP Data Model for each such Proxied Device.

The CPE Proxier is bound by the CWMP for each device it represents and MUST maintain a unique ManagementServer.ConnectionRequestURL for itself and for each device it is Proxying for. When the CPE Proxier establishes a connection to the ACS using a Virtual CWMP Device it will appear as the Proxied Device.

The Virtual CWMP Device(s) supported by the CPE Proxier MAY share the same IP address as the CPE Proxier.

J.2.1 Data Model Requirements

For the CPE Proxier the DeviceInfo.SupportedDataModel table MUST only contain entries relevant to the CPE Proxier.

For the Virtual CWMP Device representing the Proxied Device the ".DeviceInfo.SupportedDataModel" table MUST only contain entries relevant to the Proxied Device including the Device.ProxierInfo object. The Device.ProxierInfo object MUST only be represented in the data model of a Virtual CWMP Device.

The CPE Proxier supporting a Virtual CWMP Device(s) MUST provide a ManagementServer.VirtualDevice table with entries for each of the Proxied Devices it represents, regardless of the Proxy Protocol it uses to communicate with them. The ACS MAY use the ManagementServer.VirtualDevice table to identify the Proxied Device(s) that are being proxied by the CPE Proxier.

For each of the Proxied Devices supported by the CPE Proxier, the Virtual Device(s) Data Model(s) MUST contain an associated DeviceInfo.ProxierInfo Object regardless of the Proxy Protocol that connects them.

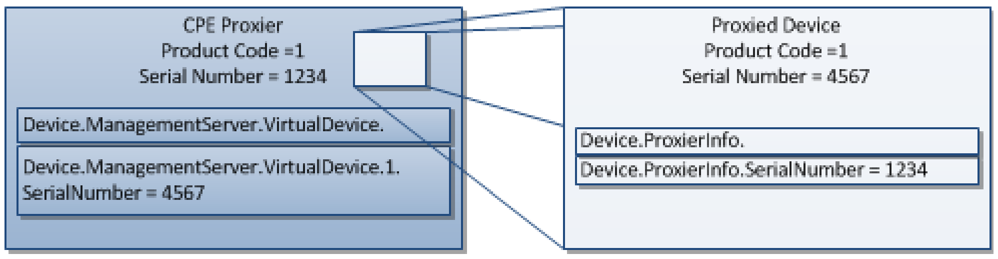


Figure 17 – CPE Proxier and Proxied Device references

J.2.2 Proxied Device Identification and Modeling

The Proxied Device MUST support or provide information required by CWMP in the DeviceInfo Object (it MUST have a means of uniquely identifying itself beyond the transport assigned address).

The CPE Proxier MUST obtain a unique OUI and Serial Number from the Proxied Device for the Virtual CWMP Device using the following options in order:

1. Use a unique OUI and Serial Number provided by the Proxied Device via the Proxy Protocol.
2. If the Proxy Protocol does not provide a unique OUI and Serial Number for the Proxied Device, the CPE Proxier MUST use the Proxied Device's MAC address to produce the OUI (defined in [39]) and set the Serial Number = MAC address.
3. If the CPE Proxier cannot obtain the Proxied Device's MAC address (or a MAC address is not provided) the CPE Proxier MUST use the unique[34] physical device identifier to provide the unique OUI and Serial Number.

When modeling a Proxied Device as a Virtual CWMP Device:

1. The Proxied Device MUST be able to be uniquely identified by its local Proxy Protocol or extensions that provide identification beyond the transport assigned addresses each time it comes online.
2. If the Proxied Device supports a connectivity stack similar to the interface stack described in TR-181 [35], this SHOULD be modeled using the interface stack.
3. The Proxied Device MUST support a Reboot mechanism.
4. The Proxied Device SHOULD support a Download mechanism, and MAY support other optional RPCs such as FactoryReset and ChangeDUState.

J.2.3 Proxied Device Availability

When a Connection Request is received for the Virtual CWMP Device and the Proxied Device is offline the CPE Proxier MUST respond with an HTTP 503 failure (see Section 3.2.2). To ensure that the ACS is contacted when the Proxied Device is once again available, the CPE Proxier MUST send a BOOT Inform via the Virtual CWMP Device when the Proxied Device comes back online.

If the CPE Proxier fails to successfully complete a CWMP Session for a Virtual CWMP Device, the Virtual CWMP Device MUST follow the session retry logic in Section 3.2.1.1

When a CPE Proxier Reboots:

- The CPE Proxier MUST send a BOOT Inform for itself.
- The CPE Proxier SHOULD NOT send BOOT Informs for any associated Virtual CWMP Devices.

If the Proxied Device communicates a Reboot to the CPE Proxier, the CPE Proxier MAY (depending upon policy) send a BOOT Inform for the Virtual Device.

[34] Since the mechanism to create the unique OUI and Serial Number from the unique physical device identifier is not defined, the same physical device may be represented with a different unique OUI and Serial Number from different CPE Proxiers.

J.3 The Embedded Object Mechanism

The Embedded Object Mechanism of providing a CWMP interface to a Proxied Device utilizes the CPE Proxiers Data Model to represent the Proxied Devices. This is done by representing the Proxied Devices as Object instances within the CPE Proxier's (Root or Service) Data Model.

J.3.1 Proxied Device Data Modeling and Provisioning

To provide visibility of which Objects in the CPE Proxier Data Model represent Proxied Devices, the CPE Proxier MUST support an EmbeddedDevice table with an entry for each supported Embedded Object. Each ManagementServer.EmbeddedDevice table entry MUST reference a row in a Multi-Instance Object that represents the Embedded Device.

The CPE Proxier MUST provide all the DT instances(s) (see Annex B/TR-106 [16]) for the Proxied Device types it supports in the DeviceInfo.SupportedDataModel table.

The CPE Proxier MUST reference the DeviceInfo.SupportedDataModel table entry(s) for each Proxied Object in the ManagementServer.EmbeddedDevice table.

For the CPE Proxier supporting multiple unique Proxied Device types within a single or multiple Proxy Protocols, each unique Proxied Device MUST be represented by at least one DT instance in the DeviceInfo.SupportedDataModel table and MAY be represented by multiple entries.

Multiple instances of an Object (all supporting the same device type) MAY be described by a single DT entry.

Guidelines in using Data Model Objects to support Proxied Devices:

1) The Proxied Devices that are not able to uniquely be identified by their local Proxy Protocol beyond the transport assigned address (such that they are not uniquely identified when removed and re-inserted into the Proxy Protocol network) MUST use Data Model Objects to provide proxy support.
2) The Proxied Devices that do not use an interface stack to model their connectivity or features SHOULD use Data Model Objects to provide proxy support.
3) The Proxied Devices that might provide no support of a reboot, factory reset and/or a download command SHOULD use Data Model Objects to provide proxy support. If such a command is needed it could be reflected in the associated Data Model, see Section I.3.4.
4) The Proxied Devices that support optional RPCs such as ChangeDUState SHOULD NOT use Data Model Objects to provide proxy support.

J.3.2 Proxied Device Availability

When the CPE Proxier receives a CWMP command for an Object or Parameter that resides on the Proxied Device, it MUST attempt to immediately execute the associated Proxy Protocol command(s) to the Proxied Device.

Prior to the CWMP Session timeout the CPE Proxier MUST return a RPC response for the command.

If the Proxy Protocol commands are not successfully responded to or applied prior to the CWMP Session timeout the CPE Proxier MUST:

- If the command was to perform configuration, the CPE Proxier MUST return a committed response (if supported by the RPC[35]).
- If the command was to retrieve information the CPE Proxier MUST return a cached result for the requested Parameter values.

Until the Proxy Protocol commands are responded to or applied (or retries exceeded), the CPE Proxier MUST continue to attempt to complete (or verify) the commands via the Proxy Protocol. While this process continues the CPE Proxier MUST return a cached result for the effected Parameters and Objects when requested.

When the command(s) are finally responded to or applied (or retries exceeded), the CPE Proxier MUST update the appropriate ManagementServer.EmbeddedDevice.{i}.LastSyncTime and/or ManagementServer.EmbeddedDevice.{i}.CommandProcessed Parameters

Depending upon the Proxy Protocol, devices that are removed from the network or are no longer available for a period of time SHOULD be removed from the Data Model.

Depending upon the Proxy Protocol when the device returns to online status (or re-discovered) if there is a unique identifier it SHOULD continue to be represented by the original Object from initial discovery. The methods to "match" this device with the Data Model Object entry are implementation dependent.

If the CPE Proxier can detect that the Proxied Device was Rebooted it MAY utilize a Data Model Parameter to mark the event. The ACS might set the Notification Attribute of the Data Model Parameter to receive notification that the Proxied Device has Rebooted.

[35] SetParameterAttributes does not allow a committed response.

Annex K. XMPP Connection Request

K.1 Introduction

The CPE WAN Management Protocol can be used to remotely manage CPE that are connected via a LAN through a Gateway. When an ACS manages a Device connected via a NAT Gateway or firewall-enabled Gateway (where the Device cannot be contacted directly by the ACS), the CPE WAN Management Protocol can still be used for management of the Device, but with the limitation that the Connection Request mechanism defined in Section 3.2.2 that allows the ACS to initiate a Session might not be usable.

The procedures defined in this Annex allow an ACS to initiate a Session with any Device, including Devices that cannot be contacted directly by the ACS. This provides the equivalent functionality of the Connection Request defined in Section 3.2.2, but makes use of a different mechanism, based on Extensible Messaging and Presence Protocol (XMPP), to accommodate this scenario.

As it relates to Devices that cannot be contacted directly by the ACS, the mechanism defined in this Annex does *not* assume that the Gateway, through which the Device is connected, supports the CPE WAN Management Protocol. This mechanism requires support only in the Device and the associated ACS.

K.2 Procedures

To accommodate the ability for an ACS to issue the equivalent of a Connection Request to a CPE, the following is required:

- The CPE MUST be able to establish a secure and authenticated connection to an XMPP Server.
- The CPE MUST be able to maintain a connection to an XMPP Server through which the XMPP Server can send unsolicited messages from an ACS-defined set of allowed addresses.

To accomplish the above items, this Annex defines a particular set of messages to be sent across a standard XMPP (as defined in RFC 6120 [40]) network. The XMPP-based Connection Request mechanism augments the existing HTTP-based Connection Request mechanism defined in Section 3.2.2. The procedures for making use of XMPP to issue a Connection Request to a CPE are summarized as follows:

1. The ACS establishes a connection to an XMPP Server.

2. The ACS enables the use of XMPP in the Device by configuring an XMPP. -Connection object, configuring the CPE's ManagementServer Object to reference the XMPP.Connection Object, and optionally configuring the set of allowed Jabber IDs within the Device's ManagementServer Object (this is shown as a SetParameterValues in the following Figure, but may also include an AddObject). This step can be omitted if the CPE is pre-provisioned with the XMPP connection values in its default settings.
3. The Device establishes an XMPP connection to the specified XMPP Server.
4. Whenever the ACS wishes to establish a connection to the CPE, it can send an XMPP Connection Request (an XMPP IQ Stanza containing a Connection Request message to its XMPP Server specifying the "to" address that matches the CPE where the Connection Request needs to be sent and a "from" address that matches one of the allowed Jabber IDs – see K.2.2.2 for details) to the XMPP Server.
5. The XMPP Server sends the XMPP IQ Stanza to the appropriate Device.
6. The Device issues an Inform request to the ACS that is specified in its ManagementServer.URL Parameter.

Note – The XMPP Connection Request mechanism is mostly described as utilizing a single XMPP Server, but there could be multiple XMPP Servers depending on the deployment. Some examples would be to deploy a single domain with multiple XMPP Servers that are clustered together or to deploy multiple domains where each domain consists of a cluster of XMPP Servers and they can all communicate to each other.

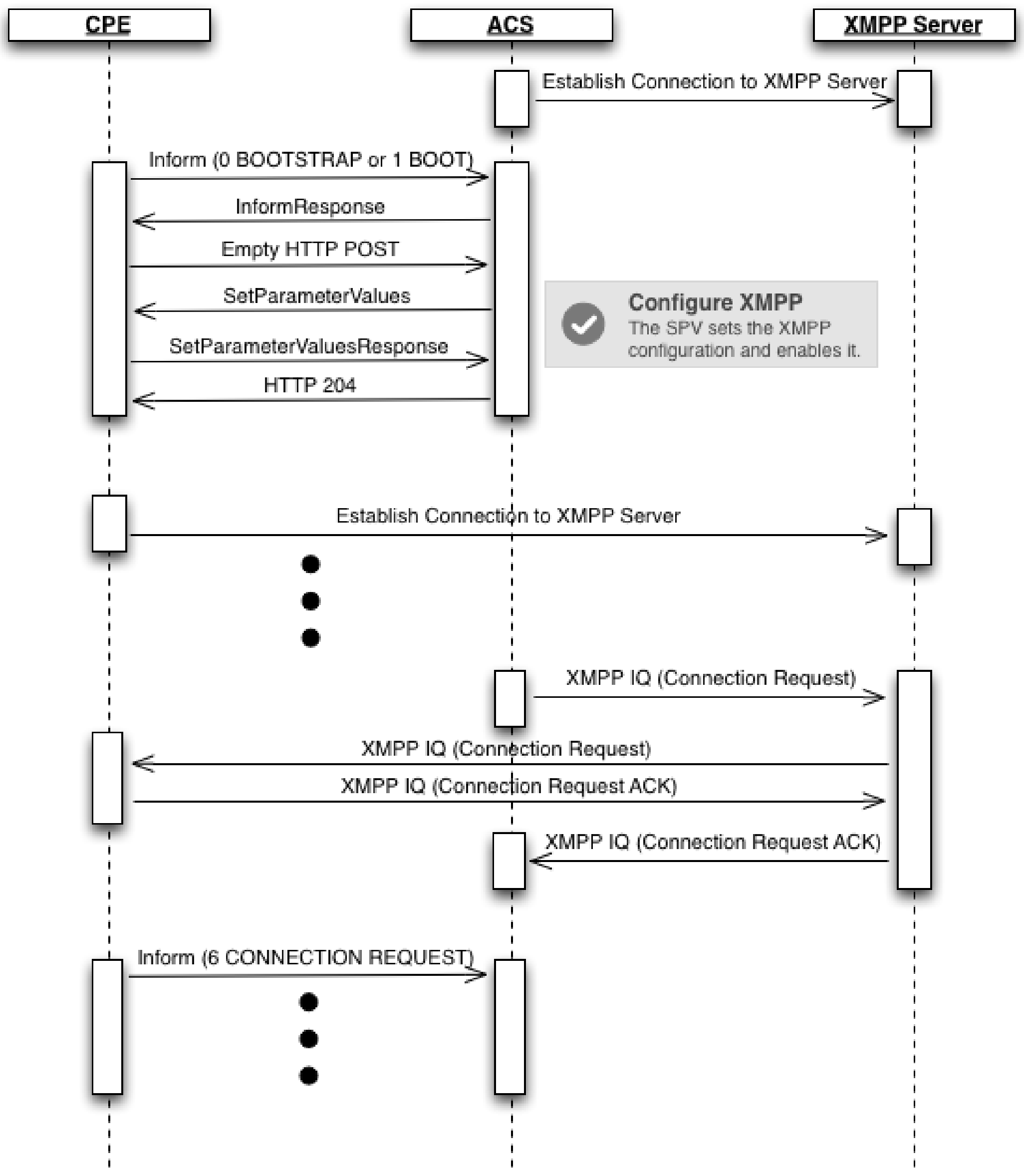

CPE
ACS
XMPP Server
Establish Connection to XMPP Server
Inform (0 BOOTSTRAP or 1 BOOT)
InformResponse
Empty HTTP POST
SetParameterValues
SetParameterValuesResponse
HTTP 204
Configure XMPP
The SPV sets the XMPP configuration and enables it.
Establish Connection to XMPP Server
XMPP IQ (Connection Request)
XMPP IQ (Connection Request)
XMPP IQ (Connection Request ACK)
XMPP IQ (Connection Request ACK)
Inform (6 CONNECTION REQUEST)

Figure 18 – XMPP Connection Request Message Sequence

K.2.1 CPE Requirements

A CPE conforming to this Annex MUST support the XMPPBasic:1 and XMPPConnReq:1 profiles as defined in tr-157-1-8.xml [41].

Whenever the ManagementServer.ConnReqXMPPConnection Parameter references an enabled instance of the XMPP.Connection table, CPE following the requirements of this Annex MUST make use of the procedures defined in XMPP (as defined in RFC 6120 [40]) to connect to the XMPP Server. The CPE MUST:

- Determine the public IP address of the XMPP Server using the rules defined in Section 3.2/RFC 6120 [40] and the recommendations described in Appendix III.4.1.

- Open an XML Stream to the XMPP Server and accept an XML Stream from the XMPP Server as defined in Section 4.2/RFC 6120 [40] and further described in Appendix III.4.2. Note: XML Streams are unidirectional and this XMPP Connection Request mechanism requires the use of two XML Streams over a single TCP connection as recommeneded in Section 4.5/RFC 6120 [40] for Client to Server sessions.
- Use TLS to establish an encrypted and secure TCP connection with the XMPP Server as defined in Section 5/RFC 6120 [40] and further described in Appendix III.4.3 (see K.2.1.1 below for XMPP Connection Request specific details).
- Use SASL to authenticate with the XMPP Server as defined in Section 6/RFC 6120 [40] and further described in Appendix III.4.4 (see K.2.1.2 below for XMPP Connection Request specific details).
- Ensure that the value of the ManagementServer.ConnReqJabberID Parameter contains the same value as the contents of the JabberID Parameter contained within the XMPP.Connection instance referenced by the ConnReqXMPP-Connection Parameter. Details on how the value of the JabberID Parameter is computed can be found in Appendix III.3.1.
- Maintain the TCP connection to the XMPP Server by utilizing the "whitespace keepalive" mechanism as defined in Section 4.6.1/RFC 6120 [40] and further described in Appendix III.5.
- Listen for XMPP Connection Request messages, and act on these messages when they arrive (see K.2.1.3 below for details).
- If the connection to the XMPP Server is ever lost, reestablish the connection as defined in Section 3.3/RFC 6120 [40] and further described in Appendix III.4.5.

Whenever the MangementServer.ConnReqXMPPConnection Parameter references an enabled instance of the XMPP.Connection table, CPE following the requirements of this Annex SHOULD:

- Establish the Connection Request XMPP connection before establishing the CWMP Session where the "1 BOOT" or "13 WAKEUP" event codes are to be delivered. In the event of a change to the ConnReqJabberID Parameter, this will allow the CPE to deliver the applicable "4 VALUE CHANGE" event code along with the "1 BOOT" or "13 WAKEUP" event code and remove the need for the CPE to immediately establish another CWMP Session for the delivery of said value change event code.

The details of these functions as they apply to the XMPP Connection Request mechanism are defined in the following Sections (details of functions that are generic to XMPP are described in Appendix III).

Note – While the CPE requirements defined here certainly apply to a Device connected via LAN to a Gateway, the same procedures can be followed by a Gateway, which might be operating behind a network-based NAT gateway. Thus the requirements are defined generically for CPE, which might be either a Device or Gateway.

K.2.1.1 XMPP Connection Encryption

When the ManagementServer.ConnReqXMPPConnection Parameter references an enabled instance of the XMPP.Connection table, CPE following the requirements of this Annex MUST negotiate a secure connection if TLS is considered "mandatory-to-negotiate" for all client-to-server XMPP connections that are used for XMPP Connection Requests (see K.3 for recommendations about XMPP Server deployments). If TLS is considered "mandatory-to-negotiate", then the XMPP connection MUST be either upgraded to a TLS encrypted XMPP channel as defined in Section 5/RFC 6120 [40] or connected over a WebSocket Secure connection as explained in Section 3.9/RFC 7395 [58] .

K.2.1.2 XMPP Channel Authentication

When the ManagementServer.ConnReqXMPPConnection Parameter references an enabled instance of the XMPP.Connection table, CPE following the requirements of this Annex MUST authenticate with the XMPP Server after establishing an XMPP connection. The XMPP connection is authenticated using the Simple Authentication and Security Layer (SASL) protocol as defined in Section 6/RFC 6120 [40]. The Username and Password parameters of the XMPP.Connection object are used as the credentials for the SASL authentication procedure.

K.2.1.3 XMPP Connection Request

A CPE conforming to this Annex MUST listen for XMPP Connection Request messages sent from an allowed list of Jabber IDs and generated by the XMPP Server specified by the XMPP.Connection instance referenced within the ManagementServer.ConnReq-XMPPConnection Parameter.

Note – a CPE MUST also continue to listen for HTTP-based Connection Requests as defined in Section 3.2.2.

The format of the XMPP Connection Request message is defined in Annex K.2.3. When the CPE receives an XMPP Connection Request message, it MUST both validate and authenticate the message.

An XMPP Connection Request message is valid if and only if the following requirements are met:

- The XMPP Connection Request message MUST be delivered within an XMPP IQ Stanza across an XML Stream that has been TLS secured according to Annex K.2.1.1 and authenticated according to Annex K.2.1.2.
- The XMPP Connection Request message MUST be well-formed XML and validated against the cwmp-xmppConnReq-1-0.xsd (http://www.broadband-forum.org/cwmp/cwmp-xmppConnReq-1-0.xsd).
- The "from" address contained within the XMPP IQ Stanza MUST match one of the addresses contained within the list-based ManagementServer.ConnReq-AllowedJabberIDs Parameter (unless the value of that Parameter is empty, which means that all addresses are allowed).

- The value of the “username” element within the “connectionRequest” element MUST match the value of the ManagementServer.ConnectionRequestUsername Parameter.

An XMPP Connection Request message is authenticated if and only if the following requirements are met:

- The value of the “password” element within the “connectionRequest” element MUST match the value of the ManagementServer.ConnectionRequestPassword Parameter.

After a CPE receives, successfully validates, authenticates, and responds to an XMPP Connection Request, it MUST establish a connection with its pre-determined ACS as described in Section 3.2.1 following the generic Connection Request requirements (as defined in 3.2.2.1) and further qualified in the following bullet points:

- The CPE SHOULD restrict the number of Connection Requests for a particular CWMP Endpoint that it accepts during a given period of time in order to further reduce the possibility of a denial of service attack. If the CPE chooses to reject a Connection Request for this reason, the CPE MUST respond to that Connection Request with the following XMPP specific error: error code 503 with a “service-unavailable” child (see K.2.3.3 for an example).
- If the CPE is already in a Session with the ACS with at least one CWMP Endpoint when it receives one or more Connection Requests, it MUST NOT terminate any Session against any CWMP Endpoint prematurely as a result. The CPE MUST instead take one of the following transport specific alternative actions:
  - Reject each Connection Request by responding with error code 503 and a “service-unavailable” child (see K.2.3.3 for an example).
  - Following the completion of the CWMP Endpoint’s current Session, initiate exactly one new Session at a time (regardless of how many Connection Requests had been received during the previous Session) in which it includes the “6 CONNECTION REQUEST” EventCode in the Inform. The Connection Requests that are not accepted MUST be rejected (with error code 503 and a “service-unavailable” child [see K.2.3.3 for an example]). If the new Session is for the CWMP Endpoint currently in Session, the CPE MUST initiate the Session immediately after the existing Session is complete and all changes from that Session have been applied.
  - If the Connection Request is not for any CWMP Endpoint currently in Session, the CPE MAY initiate a new Session with the requested CWMP Endpoint while the existing Session is still active.

This requirement holds for Connection Requests received any time during the interval that the CPE considers itself in a Session with at least one CWMP Endpoint, including the period in which the CPE is in the process of establishing the Session.

If the XMPP Connection Request can not be successfully validated or authenticated the CPE MUST return an IQ stanza with a set type attribue set to “error” according to the rules in [40] chapter 8.3 and ignore the XMPP connection Request.

K.2.2 ACS Requirements

An ACS following the requirements of this Annex SHOULD be able to enable the use of XMPP on a CPE supporting this Annex by configuring an XMPP.Connection object, configuring the ManagementServer.ConnReqXMPPConn Parameter to reference the XMPP.Connection object, and optionally configuring the ManagementServer.ConnReq-AllowedJabberIDs Parameter to contain the list of allowed Jabber IDs that can initiate an XMPP Connection Request.

An ACS conforming to this Annex MUST have access to an XMPP connection that adheres to the following requirements:

- Connects to an XMPP Server that is capable of communicating with the CPE that the ACS intends to issue a Connection Request to.
- Opens an XML Stream to the XMPP Server and accepts an XML Stream from the XMPP Server as defined in Section 4.2/RFC 6120 [40] and further described in Appendix III.4.2. Note: XML Streams are unidirectional and this XMPP Connection Request mechanism requires the use of 2 XML Streams over a single TCP connection as is recommended in Section 4.5/RFC 6120 [40] for Client to Server sessions.
- Uses TLS to establish an encrypted and secure TCP connection with the XMPP Server as defined in Section 5/RFC 6120 [40] (see K.2.2.1 below for XMPP Connection Request specific details).
- Uses SASL to authenticate with the XMPP Server as defined in Section 6/RFC 6120 [40].
- Sends XMPP Connection Request messages to a CPE supporting this Annex (see K.2.2.2 below for details).

The details of these functions as they apply to the XMPP Connection Request mechanism are defined in the following Sections.

K.2.2.1 XMPP Connection Encryption

An ACS following the requirements of this Annex MUST have access to a secure connection if TLS is considered “mandatory-to-negotiate” for all client-to-server XMPP connections that are used for XMPP Connection Requests (see K.3 for recommendations about XMPP Server deployments). If TLS is considered “mandatory-to-negotiate”, then the XMPP connection is encrypted using the STARTTLS extension of the Transport Layer Security (TLS) protocol as defined in Section 5/RFC 6120 [40].

K.2.2.2 XMPP Connection Request

An ACS conforming to this Annex MUST send XMPP Connection Request messages via the XMPP Server that the ACS has access to.

The format of the XMPP Connection Request message is defined in Annex K.2.3. An ACS MUST adhere to the following rules when sending an XMPP Connection Request message:

- The XMPP Connection Request message MUST be delivered within an XMPP IQ Stanza across an XML Stream that has been TLS secured, according to Annex K.2.2.1, and authenticated.
- The XMPP Connection Request message MUST be well-formed XML and validated against the cwmp-xmppConnReq-1-0.xsd (http://www.broadband-forum.org/cwmp/cwmp-xmppConnReq-1-0.xsd).
- The “from” address contained within the XMPP IQ Stanza MUST match one of the addresses contained within the list-based ManagementServer. -ConnReqAllowedJabberIDs Parameter (unless the value of that Parameter is empty, which means that all addresses are allowed).
- The value of the “username” element within the “connectionRequest” element MUST match the value of theManagementServer.ConnectionRequestUsername Parameter for the CPE where the XMPP Connection Request is being sent.
- The value of the “password” element within the “connectionRequest” element MUST match the value of the ManagementServer.ConnectionRequestPassword Parameter for the CPE where the XMPP Connection Request is being sent.

K.2.3 Message Flows

The following figures show example message flows associated with the procedures defined in Annex K.2.1 and K.2.2 to support Connection Requests to Devices.

In all of the examples, the address uses the notation *(L@D/R)*, where *L* is the local-part (XMPP. Connection.{i}.Username), *D* is the domain-part (XMPP. Connection.{i}.-Domain), and *R* is the resource-part (XMPP. Connection.{i}.Resource), where XMPP.Connection.{i} is the instance referenced by the ManagementServer.ConnReq-XMPPConn Parameter.

K.2.3.1 Connection Request

The Connection Request message is formatted within an XMPP IQ Stanza and is defined with the Broadband Forum connectionRequest element as seen in Figure 19. This element utilizes the existing ConnectionRequestUsername and ConnectionRequest-Password Parameters contained in the ManagementServer object of the CPE where the XMPP Connection Request message is addressed. As the XMPP connection is an encrypted and authenticated channel, there is no need for these values to be further encrypted.

```
<iq from="[ACS-identity]" to="L@D/R" id="cr001" type="get">
  <connectionRequest xmlns="urn:broadband-forum-org:cwmp:xmppConnReq-1-0">
    <username>username</username>
    <password>password</password>
  </connectionRequest>
</iq>
```

Figure 19 – Connection Request: sent from ACS

K.2.3.2 Connection Request Successful Response

A successful response from the XMPP client contained within the CPE is an empty result IQ Stanza to the ACS as seen in Figure 20.

```
<iq from="L@D/R" to="[ACS-identity]" id="cr001" type="result" />
```

Figure 20 – Successful Response to Connection Request: sent from CPE

K.2.3.3 Connection Request Error Response

An unsuccessful response is an error IQ stanza with the associated error type element.

If the CPE is not available, the response from the XMPP Server MUST contain a <service-unavailable/> error as seen in Figure 21.

If the CPE does not support the "urn:broadband-forum-org:cwmp:xmppConnReq-1-0" namespace, the response from the CPE MUST contain a <service-unavailable/> error as seen in Figure 21.

```
<iq from="L@D/R" to="[ACS-identity]" id="cr001" type="error">
  <error type="cancel">
    <service-unavailable xmlns="urn:ietf:params:xml:ns:xmpp-stanzas" />
  </error>
</iq>
```

Figure 21 – Type Cancel Error Response to Connection Request from ACS

If the XMPP Connection Request cannot be authenticated, the CPE MUST send a <not-authorized> error as seen in Figure 22.

```

<iq from="L@D/R" to="[ACS-identity]" id="cr001" type="error">
  <connectionRequest xmlns="urn:broadband-forum-org:cwmp:xmppConnReq-1-0">
    <username>username</username>
    <password>password</password>
  </connectionRequest>
  <error type="cancel">
    <not-authorized xmlns="urn:ietf:params:xml:ns:xmpp-stanzas" />
  </error>
</iq>
```

Figure 22 – Authentication Error Response to Connection Request from ACS

Note – All rules defined in 8.3.1 of RFC 6120 [40] apply to the creation of the error stanza. More specifically, Rule 6 in 8.3.1 of RFC 6120 [40] maintains that it is up to the implementation as to whether or not the original XML is included in the error stanza, which means that both Figure 21 (orignal XML is omitted) and Figure 22 (original XML is included) are valid..

K.3 XMPP Server Deployment Requirements

An XMPP Server being used to send XMPP Connection Request messages between an ACS and a CPE that conform to this Annex MUST adhere to the requirements defined in XMPP (as defined in RFC 6120 [40]). In addition, it is RECOMMENDED that the XMPP Servers be configured as follows:

- TLS is "mandatory-to-negotiate" for all client-to-server communications.
- TLS is "mandatory-to-negotiate" for all server-to-server communications.

The configuration recommendations are for environments where the XMPP Servers are not fully under the control of the Service Provider, which would imply that they are potentially at risk for security attacks.

K.4 Security Considerations

The following security considerations associated with the procedures defined in this Annex are identified:

- The XMPP specification describes several security considerations. The reader is referred to Section 13 of RFC 6120 [40] for a detailed description of these security considerations and the associated risk.
- Section 13.6 of RFC 6120 [40] requires that XMPP Client and Server implementations support "strong security", which refers to the use of security technologies that provide both mutual authentication and integrity checking, but it is up to the installed configuration to ensure that these security details are enabled. Mutual authentication is of particular importance for a secure XMPP Connection Request mechanism especially for the server-to-server communications.
- In order for the XMPP Connection Request to be considered secure, the ACS MUST configure the allowed list of Jabber ID from addresses (the ManagementServer.ConnReqAllowedJabberIDs Parameter) and the XMPP Connection Request authentication credentials (the ConnectionRequestUsername and ConnectionRequestPassword Parameters on the ManagementServer Object) during an SSL/TLS secured CWMP Session.

Annex L. CPE standby-related behaviors

L.1 Introduction

Some CPE have the capability to enter a standby state where some of their functions are not available or not available immediately. The decision to enter or leave standby is independent from CWMP, but the CPE has to behave in a consistent way when dealing with the ACS. This Annex describes how to achieve this.

The relevant use cases are:

a. If a CPE goes into standby, misses at least one Scheduled or Periodic Contact with the ACS, then wakes up, the ACS has a way to be informed of the fact that the CPE is available again. The ACS has some control on the timeliness of that information. The ACS is also able to decide if it is interested in the information or not.

b. If a CPE goes into standby and then wakes up after the ACS failed to get a response to a Connection Request, the ACS has a way to be informed of the fact that the CPE is available again. The ACS has some control on the timeliness of that information. The ACS is also able to decide it is not interested in that information.

c. If a CPE goes into standby and none of the above situations is true, there is no need to inform the ACS when the CPE is available again, because it has not been Seen Missing.

d. If, for some reason, the ACS would not want the CPE to miss any Periodic Contact for a period of time, the ACS is able to determine whether the CPE offers control on that behavior and, if so, to configure it as necessary. Whenever the ACS requests a Scheduled Contact, it does not want the CPE to miss it.

e. If, for some reason, the ACS would not want the CPE to be unreachable by Connection Requests for a period of time, the ACS is able to determine whether the CPE offers control on that behavior and, if so, to configure it as wanted.

f. The ACS is able to distinguish between a CPE that reboots and a CPE that wakes up from standby.

g. Standby modes are implemented to improve energy efficiency; their use does not result in an increased activity of the CPE and/or the ACS.

L.2 Procedures

L.2.1 Use of WAKEUP Event

A specific Event allows the CPE to indicate to the ACS that it is sending an Inform as a result of waking up from a standby state. This Event is “13 WAKEUP”. This addresses *use case f.*

A CPE SHOULD NOT issue a BOOT Event when a WAKEUP Event is appropriate, because a BOOT Event is likely to trigger a heavier procedure on the ACS, which would go against *use case g*.

L.2.2 Conditions requiring a WAKEUP Event

Three parameters are defined in the data model as policy elements that can be set by the ACS. They allow the CPE to decide when it has to send a WAKEUP Event. (For those which are thresholds, a negative value indicates the CPE MUST NOT issue a WAKEUP for exceeding that threshold.)

- A CPE waking up from a **non CR-Aware Standby** that lasted more than CRUnawarenessMaxDuration MUST issue a WAKEUP Event (*use case b*).
- A CPE waking up from a **non fully Timer-Aware Standby** that made it miss more than MaxMissedPeriodic Periodic Contacts MUST issue a WAKEUP Event (*use case a*).
- A CPE waking up from a **non fully Timer-Aware Standby** that made it miss at least one Scheduled Contact MUST issue a WAKEUP Event if NotifyMissedScheduled is true (*use case a*).
- A CPE which is not in one of the above situations MUST NOT issue a WAKEUP Event (*use case c*).

L.2.3 When to deliver a WAKEUP Event

The CPE MUST deliver the WAKEUP Event as soon as one of the conditions in L.2.2 is met, issuing a specific Inform for that purpose if needed. If several conditions apply, only one event summarizing these conditions is sent.

L.2.4 Management of standby modes by the ACS

Two capability parameters allow the CPE to advertise the fact that it is able to wake up based on network activity or internal timers. Three policy elements allow the ACS to require the CPE to remain in CR-Aware Standby or to honor Periodic or Scheduled Contacts:

- If NetworkAwarenessCapable is true and the ACS sets CRAwarenessRequested to true, the CPE MUST NOT go into a non CR-Aware Standby state (*use case e*).
- If SelfTimerCapable is true and the ACS sets PeriodicAwarenessRequested to true, the CPE MUST NOT miss any Periodic Contact (*use case d*).

- If ScheduledAwarenessRequested is true, the CPE MUST NOT miss any Scheduled Contact (*use case d*). This holds even if SelfTimerCapable is false; in that case the CPE will not go into standby at all until all Scheduled Contacts have been honored.

Annex M. UDP Lightweight Notification

M.1 Introduction

The procedures defined in this Annex extend the CPE WAN Management Protocol to allow for a UDP-based notification mechanism. This "lightweight" mechanism is only applicable if the service provider does not require reliability of packet delivery. In general, UDP usage in IP networks is reliable, i.e. the packet loss probability is low. Service providers might want to use this method for historical trend analysis of non-sensitive data.

This UDP lightweight notification mechanism is not intended as a replacement of the existing Inform mechanism but as an alternative to the current Active and Passive notification mechanism.

M.2 Procedures

M.2.1 CPE requirements

A CPE conforming to this Annex indicates support of the UDP lightweight notification mechanism by including "UDP" in the capability Parameter ".ManagementServer.LightweightNotificationProtocolsSupported". If this Parameter is missing or doesn't include the "UDP" enumeration the mechanism described in this Annex is not supported by the CPE.

CPE supporting this mechanism MUST send the UDPLightweightNotification message whenever a change occurs in the value of a parameter (or parameters), that the ACS has marked for "Active lightweight notification" (value 5 or 6) via the SetParameterAttributes method, by an external cause (a cause other than the ACS itself).

Whenever a change occurs in the value of a parameter (or parameters), that the ACS has marked for "Passive lightweight notification" (value 3 or 4) via the SetParameterAttributes method, by an external cause (a cause other than the ACS itself), the CPE MUST send the new value either with the next UDPLightweightNotification message sent for an active lightweight notification parameter or at the latest when the next lightweight notification trigger interval (as dictated by the .ManagementServer.LightweightNotificationTriggerInterval and .ManagementServer.LightweightNotificationTriggerTime parameters) expires. If the lightweight notification trigger interval is disabled, a new Passive Lightweight notification is only sent with the next UDPLightweightNotification message for an active lightweight notification parameter.

The CPE MUST include the new value (or values) as a ParameterStruct in the associated UDPLightweightNotification message. The calling arguments for this message are described in Table 108 below.

A UDPLightweightNotification message MUST be sent to the appropriate address as defined by ManagementServer Parameters. The host name or IP address is extracted from the ManagementServer.UDPLightweightNotificationHost Parameter, and the port number is taken from the ManagementServer.UDPLightweightNotificationPort Parameter. If these Parameters are not implemented then the destination host MUST be be the ACS (using the pre-determined ACS address - see section 3.1), and the destination port MUST be 7547, which has been assigned by IANA for the CPE WAN Management Protocol (see [20]).

The message MUST be transmitted within a single UDP packet.

Note – The UDP packet includes several headers that are not under the direct control of the application, i.e. HTTP headers, IP header (IPv4 or IPv6) and UDP header. Also, the application cannot know whether the packet will be fragmented somewhere between its source and destination. Therefore, for a given deployment, it might be necessary to apply a heuristic algorithm when determining the maximum message length.

M.2.2 ACS requirements

The ACS MUST ignore a UDPLightweightNotification that is not successfully validated and authenticated.

A UDPLightweightNotification message is validated if and only if it conforms to the requirements in M.2.3 and the following requirements are met:

- The Timestamp MUST be greater than or equal to the Timestamp value for the UDP Lightweight Notification message that had been most recently received, validated, and authenticated.
- The Username MUST match the value of the ManagementServer.Username parameter.

A UDPLightweightNotification message is authenticated if and only if the following requirements are met:

- The Signature given by the value of the “Signature” HTTP header MUST match the value of the signature locally computed by the ACS following the procedure specified in section M.2.3 using the local value of the ManagementServer.Password Parameter.

The following additional requirements relate to Timestamp processing:

- To allow the above comparison to be made, the ACS MUST maintain a persistent record of the Timestamp value of the most recent UDP Lightweight Notification that was successfully validated and authenticated. The Timestamp value for any UDP Lightweight Notification message that fails to be validated or authenticated MUST NOT be recorded.
- The ACS MAY place stricter requirements on the Timestamp than stated above. The ACS MAY, for example, additionally verify that the Timestamp is within a

time window relative to its understanding of the current time. If a ACS chooses to do this, it SHOULD avoid making the time window too narrow, in order to allow for a reasonable margin of error in both the ACS and CPE.

M.2.3 UDPLightweightNotification message

The UDPLightweightNotification message MUST conform to the following requirements:

- It MUST be a valid HTTP 1.1 POST message as defined in [6], [7].
- The HTTP message body MUST contain a valid XML structure containing the message arguments defined in M.2.4 coded according to the defined schema (see M.2.5).
- The request URL MUST be set to the root path "/".
- The message MUST include the following HTTP headers, and any other HTTP headers MUST NOT be included:
    - Host
    - Content-Type (with the value of "text/xml; charset=utf-8")
    - Content-Length
    - Signature
- The signature header contains the 40-character hexadecimal representation (case insensitive) of HMAC-<digest function> (Key, Text) [22] where:
    - "<digest-function>" is the digest function configured with the parameter "Device.ManagementServer.LightweightDigest". If this parameter is empty or missing, the function defaults to SHA1
    - "Key" is the value of the parameter ManagementServer.Password as read from the CPE.
    - "Text" is the whole message body (i.e., not including the headers or empty line), treated as a sequence of octets.
- The message body MUST be UTF-8 encoded.

M.2.4 Message arguments

The calling arguments for the UDPLightweightNotification message are defined in Table 108. There is no response message associated with the message.

Table 108 – UDPLightweightNotification arguments

Argument	Type	Value
TS	int	Timestamp. The number of seconds since the Unix epoch until the time the message is created (the standard Unix timestamp).
UN	string(256)	Username: The value of the parameter ManagementServer.Username as read from the CPE.
CN	string	Cnonce: A random string chosen by the CPE.
OUI	string(6)	Organizationally unique identifier of the device manufacturer. Represented as a six hexadecimal-digit value using all upper-case letters and including any leading zeros. The value MUST be a valid OUI as defined in [13]. The value MUST be the same as the value of the DeviceInfo.ManufacturerOUI Parameter.
ProductClass	string(64)	Identifier of the class of product for which the serial number applies. That is, for a given manufacturer, this Parameter is used to identify the product or class of product over which the SerialNumber Parameter is unique. The value MUST be the same as the value of the DeviceInfo.ProductClass Parameter.
SerialNumber	string(64)	Identifier of the particular device that is unique for the indicated class of product and manufacturer. The value MUST be the same as the value of the DeviceInfo.SerialNumber Parameter.
Param	ParameterStruct…	A structure with the name and value of the parameters notified, defined in Table 109. This element can be repeated *n* times, depending on the number of parameters to be notified in the same message. There MUST be at least one structure in the message, but there can be up to *n* structures in the same message until the maximum length of the message is reached.

Table 109 – ParameterStruct definition

Argument	Type	Value
Name	string(256)	This is the name of a Parameter. The CPE MUST treat the Parameter name as case sensitive.
Value	anySimpleType	This is the value of the Parameter. The CPE MUST treat string-valued Parameter values as case-sensitive. The rules regarding the anySimpleType data type, contained in Table 12, MUST be applied..

Note - The defined values are derived from the ParameterValueStruct structure in Table 17.

M.2.5 UDPLightweightNotification XML Schema

The XML schema, which is the normative definition of the UDPLightweightNotification message format, is specified in the referenced file below:

Version	Namespace	XSD
1.0	urn:broadband-forum-org:cwmp:lwnotif-1-0	http://www.broadband-forum.org/cwmp/UDPLightweightNotification-1-0.xsd

M.2.6 UDPLightweightNotification example

As explained in M.2.4 the UDPLightweightNotification message is sent using the HTTP POST message. An example is given below.

```
POST / HTTP/1.1
Host: acs.server.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
Signature: 42A424FB0C4938BBF08259514C64ABFC1E74C61F

<?xml version="1.0" encoding="UTF-8"?>
<Notification xmlns="urn:broadband-forum-org:cwmp:lwnotif-1-0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:broadband-forum-org:cwmp:lwnotif-1-0 http://www.broadband-
                      forum.org/cwmp/¬cwmp-UDPLightweightNotification-1-0.xsd">
    <TS>1120673700</TS>
    <UN>ACS57689</UN>
    <CN>XTGRWIPC6D3IPXS3</CN>
    <OUI>00D09E</OUI>
    <ProductClass></ProductClass>
    <SerialNumber>134SS03013</SerialNumber>
    <Param>
        <Name>Device.ManagementServer.PeriodicInformEnable</Name>
        <Value xsi:type="xs:boolean">false</Value>
    </Param>
    <Param>
        <Name>Device.ManagementServer.PeriodicInformInterval</Name>
        <Value xsi:type="xs:unsignedInt">60</Value>
    </Param>
</Notification>
```

Annex N. HTTP Bulk Data Collection

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

N.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.

N.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".

N.2.1 Use of the URI Query Parameters

The HTTP Bulk Data transfer mechanism allows fully qualified parameters (a fully qualified parameter is one whose parameter path does not include any wildcards nor is a partial path) 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"

N.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.

N.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.

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 110. The CPE MUST choose the wait interval by randomly selecting a number of seconds from the range given by the data transfer retry count. If the Device.BulkData.Profile.{i}.HTTP.PersistAcrossReboot Parameter indicates that the failed data transfers are persisted across reboots, then upon 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[36]	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 110. The CPE MUST continue to retry a failed data transfer until it

[36] These symbols are used in Table 110 – Data Transfer Retry Wait IntervalsTable 3.

is successfully terminated or until the next reporting interval for the data transfer becomes effective. Section N.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 110 – 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
#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}

N.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 RFC 5246 [14] (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 [50] 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 [49], 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.

N.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 [52] and for JSON the media type is application/json as specified in RFC 7159 [53]. 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.

N.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"

N.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).

N.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.
```

N.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.

N.3.4 Encoding of CSV Bulk Data

CSV Bulk Data SHOULD be encoded as per RFC 4180 [52] 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 N.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 Section N.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="&#13;&#10;"
- .BulkData.Profile.1.CSVEncoding.EscapeCharacter="&quot;"

N.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 N.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 Section N.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"

Note - 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 [16].

N.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.

N.3.5 Encoding of JSON Bulk Data

Using the HTTP example in Section N.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"

N.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 N.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.

N.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}
      ]
```

N.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.

N.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.

N.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 [16]. The encoding of JSON Data Types MUST adhere to RFC 7159 [53].

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 111 – 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.
int, long, unsignedInt, unsignedLong	Number
string	String

N.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.

N.4.1 CSV Encoded Report Examples

N.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="&#13;&#10;"
- .BulkData.Profile.1.CSVEncoding.EscapeCharacter="&quot;"
- .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,unsignedL ong

N.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="&#13;&#10;"
- .BulkData.Profile.1.CSVEncoding.EscapeCharacter="&quot;"
- .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

N.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
    }
  ]
}
```

Annex O. HEARTBEAT Event Procedures

O.1 Introduction

In certain deployments and situations, CPEs are requested to contact the ACS on a very frequent basis for the purposes of informing the ACS that the CPE is still active and alive. This type of contact is called a HEARTBEAT Event. A HEARTBEAT Event differs from a PERIODIC Event in that the HEARTBEAT Event contains only the Forced Inform parameters and no other Events are emitted in the same Inform session as the additional processing associated with other Events would cause undue load on an ACS. The HEARTBEAT Event is emitted by a CPE to the ACS using the Inform method when the conditions of the CPE's Heartbeat policy are fulfilled.

O.2 Procedures

O.2.1 Use of HEARTBEAT Event

The HEARTBEAT Event allows the CPE to indicate to the ACS that it is sending an Inform RPC to indicate that the CPE is active/alive.

A CPE MUST NOT issue another Event in the same Inform session when a HEARTBEAT Event is issued. This is because HEARTBEAT Events, when enabled, are expected to be frequent and other Events in the same Inform session will likely trigger a heavier procedure on the ACS.

O.2.2 Interactions with active Inform Sessions

If a HEARTBEAT Event is triggered when an Inform Session is currently active, then the CPE MUST wait until the current Inform Session has completed prior to issuing a new Inform session for the HEARTBEAT Event.

If a HEARTBEAT Event is triggered at the same time another Event can trigger an Inform Session, the Inform Session for the HEARTBEAT Event MUST take precedence over the Events. As such, the HEARTBEAT Event will establish its Inform Session and then the other Events will establish their Inform Session after the completion of the HEARTBEAT Event Inform Session.

If a HEARTBEAT Event is triggered at the same time the another HEARTBEAT Event Inform session is active, the CPE MUST discard the new HEARTBEAT Event.

O.2.3 Interactions with retrying Inform Sessions

If a HEARTBEAT Event is outstanding or triggered while an Non-HEARTBEAT Inform is being retried, the CPE MUST wait and allow the Non-HEARTBEAT Inform to establish an Inform Session. Once the Inform Session for the Non-HEARTBEAT Inform is completed, a new Inform Session for the HEARTBEAT Event MUST be initiated.

If a HEARTBEAT Event is triggered while an Inform Session for a previous HEARTBEAT Event is being retried, the CPE MUST discard the new HEARTBEAT Event.

O.2.4 Conditions requiring a HEARTBEAT Event

A HEARTBEAT Event is triggered based on the conditions represented by the objects and parameters defined in the TR-181i2 data model's HeartbeatPolicy:1.0 profile [35].

A CPE that implements the HEARTBEAT Event MUST implemenent the HeartbeatPolicy:1 profile.

Appendix I. CPE Proxier Implementation Guidelines

I.1 Introduction

This Appendix suggests possible implementation guidelines and mechanisms to assist a CPE Proxier to provide CMWP Proxy Management of a Proxied Device connected via a Proxy Protocol. Proxy Protocols supported by a CPE Proxier could be synchronous, asynchronous, IP based, non-IP based, standard, or proprietary.

The CPE Proxier that supports a Proxy Protocol(s) supports the underlying communications protocol and capabilities to communicate via the Proxy Protocol. The CPE Proxier also supports the mechanisms needed to discover and manage the Proxied Devices via the Proxy Protocol.

To provide support for a varying number of Proxy Protocols this Appendix suggests mechanisms to enable this. Some paradigms are shared by both the Virtual CWMP Device and Embedded Object Mechanisms; these are covered in Section I.2. For guidelines specific to Embedded Object Mechanism see Section I.3, for the Virtual CWMP Device see Section I.4. Section I.5 is used to provide an example of how the Data Model extensions mentioned in Annex J are used.

I.2 Common Guidelines for the Virtual CWMP Device and Embedded Object Mechanisms

All of the guidelines in this section apply to both the Virtual CWMP Device and the Embedded Object Mechanisms proxying of CWMP RPC commands.

I.2.1 Unsupported CWMP RPC Commands by the Proxy Protocol

The CWMP supports an AccessList attribute that might not be supported by a Proxy Protocol. If the Proxy Protocols cannot support such a mechanism, and the ACS attempts to modify the AccessList attribute to any other value, the CPE could choose to reject the modification using a SOAP fault '9001' Request denied.

I.2.2 Support for Proxy Protocol Methods with no Matching RPC

Proxy Protocols might use a separate method call for various functions such as Ping, TraceRoute, and GetDeviceStatus etc. For these operations CWMP uses the Data Model to perform such functions (not an RPC). The suggested solution is to have the CPE

Proxier use the Data Model (as defined in the DT instance(s); see Annex B/TR-106 [16]) to model the particular method.

Here is an example of how a CWMP TraceRoute Diagnostics test that is defined in [35] would map to the UPnP DM TraceRoute Service:

1. The ACS via CWMP SetParameterValues sets the appropriate Parameter values for test setup in the TraceRoute Object, and then sets the TraceRoute.DiagnosticsState to Requested.
2. Next the CPE Proxier initiates a UPnP TraceRoute service request to the Proxied Device, using the appropriate Parameters from the TraceRoute Object.
3. The CPE Proxier receives notification of the completion of the TraceRoute diagnostic (either through UPnP notifications from the Proxied Device or polling the Proxied Device).
4. Using the GetTraceRouteResult service the CPE Proxier retrieves the TraceRoute diagnostic results from the Proxied Device. These results are stored in the CPE Proxier for future retrieval from the ACS.
5. The CPE Proxier will send a CWMP Event "8 DIAGNOSTICS COMPLETE" in a CWMP Inform to the ACS upon the completion of the test.

I.2.3 Support for Transactional Integrity

To provide the transactional integrity requirement in Section 3.7.1.1, during the course of a Session the CPE Proxier might need to return the requested value of any Parameter that was set previously within the same Session. The CPE Proxier could use the requested value from a previous configuration command in the same Session, rather than retrieving a new (and possibly modified) value for the later request within the same CWMP Session.

I.3 Embedded Object Mechanism

The following guidelines are for a CPE Proxier when using the Embedded Object Mechanism.

I.3.1 Device Discovery

Proxy Protocols have various mechanisms for discovering devices. Some require polling or listening on a particular TCP port, others might need an API to the supported Proxy Protocols.

When using the Embedded Object Mechanism the CPE Proxier will only support a Proxied Device that is represented in a DeviceInfo.SupportedDataModel DT entry(s).

When a Device is initially discovered by the CPE Proxier the following steps occur.

- Retrieve the information from the discovered device, via the Proxy Protocol, that is necessary to match and verify that the discovered device is supported by the CPE Proxier.

- If the discovered device is supported, The CPE Proxier could populate the appropriate Data Model Object and Parameters based on the information retrieved from the device utilizing the appropriate methods of the Proxy Protocol.
- The CPE Proxier would then update the ManagementServer.EmbeddedDevice table to reference the DeviceInfo.SupportedDataModel table DT entry for the Proxied Device and also reference the newly created Data Model Object.

For the ACS to discover the device, it can utilize the "ManagementServer.-EmbeddedDeviceNumberOfEntries" Parameter or the number of entries Parameter for the particular Object representing the device (possibly setting a change notification on either Parameter for notification of the newly discovered Proxied Device).

I.3.2 **CPE Proxier use of Polling**

Polling could be used by the CPE Proxier to support change notifications for Parameters on the Proxied Device, if a Proxy Protocol does not support such a mechanism.

The CPE Proxier might also utilize polling of Parameter values on the Proxy Protocol to keep the cached Objects and Parameters 'in sync' with the Proxied Device, and upon change send the appropriate CWMP notification.

I.3.3 **ACS Query of RPC Execution**

When the ACS requests Parameter values from the CPE Proxier it could query the ManagementServer.EmbeddedDevice.{i}.LastSyncTime Parameter to insure the validity of the data. When the ACS receives a committed response from the CPE Proxier for a configuration command, it could request the "ManagementServer.EmbeddedDevice.-{i}.CommandProcessed" Parameter to verify the state of the command.

I.3.4 **Support for Proxy Protocol Methods Reboot and Download**

For a Proxied Device that might not fit the criteria for a Virtual CWMP Device, yet supports a Reboot command or Download feature, an Embedded Object mechanism can be used. The methods can be supported via Data Model Objects and Parameters that function similar to the TraceRoute example above (Section I.2.2) and other Data Model mechanisms (as defined in the DT instance(s); see Annex B/TR-106 [16]).

The Download feature for example, which would typically be used to perform a firmware upgrade, could be modeled within the object that represents the Embedded Device. The following **Parameters** could be used to perform a Download to that Proxied Device:

URL: This is where the file to be downloaded resides.

Username: The username to be used when accessing the URL.

Password: The password to be used when accessing the URL.

StartDownload: A command parameter that when set to true causes the download to begin.

Status: The results of the download represented as an enumerated string (NOT_STARTED, PENDING, SUCCESS, FAILED).

ErrorMessage: An empty string unless Status is FAILED, in which case this would be a description of why the Download failed.

I.4 Virtual CWMP Device Mechanism

The following guidelines are for a CPE Proxier when using the Virtual CWMP Device Mechanism.

I.4.1 Device Discovery

Proxy Protocols have various mechanisms for discovering devices. Some require polling or listening on a particular TCP port, others might need an API to the supported Proxy Protocols.

The Virtual CPE Device reflects the Data Model it supports in its own DeviceInfo.SupportedDataModel DT entry.

When a Device is initially discovered by the CPE Proxier the following steps occur.

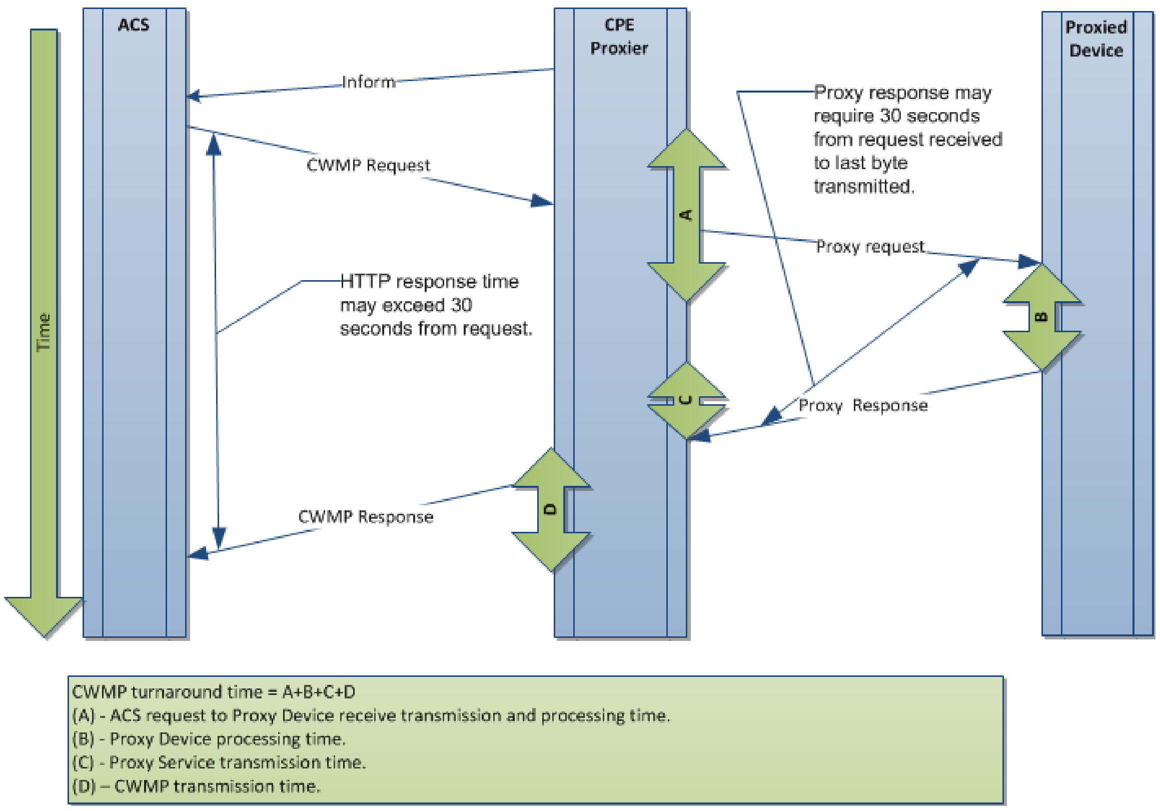
- Retrieve the information from the discovered device, via the Proxy Protocol, that is necessary to match and verify that the discovered device is supported by the CPE Proxier.
- When a CPE Proxier first discovers a device that it supports, the CPE Proxier represents the Proxied Device and connects to the ACS (as any new device would when it connects to the ACS for the first time) by sending a BOOTSTRAP event (described in Table 39).
- Update the ManagementServer.VirtualDevice table in the CPE Proxier.

I.4.2 Request for Session Timeout Extension

One aspect of all Proxy Protocols is the turnaround time. A single Proxy Protocol command might exceed the CMWP turnaround time. A synchronous CWMP command could also result in multiple Proxy Protocol transactions or asynchronous transactions to complete, which increases the number of operations subject to the Proxy Protocol timeout within the CWMP Session timeout.

Figure 23 is an example of how the CWMP turnaround time might exceed the CWMP Session timeout due to the additional Proxy Protocol processing time.

To provide an indication to this condition, the CPE Proxier has the option of notifying the ACS with the "SessionTimeout" Inform SOAP header (defined in Table 4) requesting to extend the Session timeout. Additionally a Virtual CWMP Device would use the "SessionTimeout" value in cases where an operation failed to complete in the previous Session due to a timeout.


ACS
CPE Proxier
Proxied Device
Time
Inform
CWMP Request
HTTP response time may exceed 30 seconds from request.
Proxy response may require 30 seconds from request received to last byte transmitted.
Proxy request
Proxy Response
CWMP Response
A
B
C
D

CWMP turnaround time = A+B+C+D
(A) - ACS request to Proxy Device receive transmission and processing time.
(B) - Proxy Device processing time.
(C) - Proxy Service transmission time.
(D) – CWMP transmission time.

Figure 23 – Turnaround time

I.4.3 CPE Proxier use of Caching

Caching by the CPE Proxier can provide a mechanism to store information and provide CWMP services (Attributes, Notifications) that are otherwise not supported by the Proxy Protocol.

Below are CWMP mechanisms that might use caching and polling in order to support a Proxy Protocol mechanism that cannot be relayed. It is not suggested to use the cache for returning values and setting values to the Proxied Device without attempting to reach the device first. CWMP Proxying is predicated on the relaying of commands within the CWMP Session as shown in Figure 23.

- **Device discovery information**

  If the CPE Proxier has the capability to retain Proxied Device specific information such that when the device is rediscovered (possibly with a new network address) it will appear to be the same Device as when it was initially discovered, the ACS will be able to manage the Proxied Device as the same device instead of managing it as a new device.

- **Single CWMP RPC results in multiple Proxy Protocol RPCs**

  Caching might be used in the case where a single CMWP RPC results in multiple Proxy Protocol method calls. This allows the CPE Proxier to retain information from each call and supply the single RPC response when finished.

For example, a GetParameterNames RPC call with NextLevel=false would result in two UPnP DM commands GetSupportedParameters and GetInstances to respond to the single GetParamenterNames call.

Another example would be a SetParameterValues RPC call to a CPE Proxier that supports an asynchronous Proxy Protocol that requires a set command and a separate get command to verify the set operation is complete.

- **Support for unsupported CWMP mechanisms**

  A Proxy Protocol that does not support Attributes and Notifications might need the CPE Proxier to utilize caching of the Attributes and Notification requests.

  The CPE Proxier might use polling to keep cached Objects and Parameters 'in sync' with the Proxied Device, and upon change send the appropriate CWMP notification.

  A Proxy Protocol that does not have a matching 'Reboot' command might need the CPE Proxier to utilize similar Proxy Protocol commands to interpret the Reboot request.

I.4.4 Virtual CWMP Device Error Scenarios

The Virtual CWMP Device utilizes connectivity between the CPE Proxier and the ACS, and the CPE Proxier and the Proxied Device (see Figure 16). In this situation there are two communication links that are needed to complete the RPC operation. Failure of either of these communication links might result in a CMWP Session fault, and will require the Virtual CWMP Device to follow the session retry logic in Section 3.2.1.1.

I.5 Proxy Management Support Example

This is an example of a CPE Proxier that incorporates a Virtual CWMP Device Mechanism and Embedded Object Mechanism (utilizing Root and Service Objects). The following figure describes a Router that is also a CPE Proxier device that supports multiple Proxied Devices.

The NAS 1 and 2 are Proxied Devices that are supported by the Router device via the Virtual CWMP Device Mechanism.

Utilizing the Embedded Object Mechanism the Router device supports the Meter Devices 1 and 2 via a Proxied Device table “Device.Meter.{i}.” with 2 entries that exist in the Router Data Model. The Router supports IP cameras 1 and 2 as Proxied Devices utilizing the “Device.Services.IPCamera.{i}” service Objects.

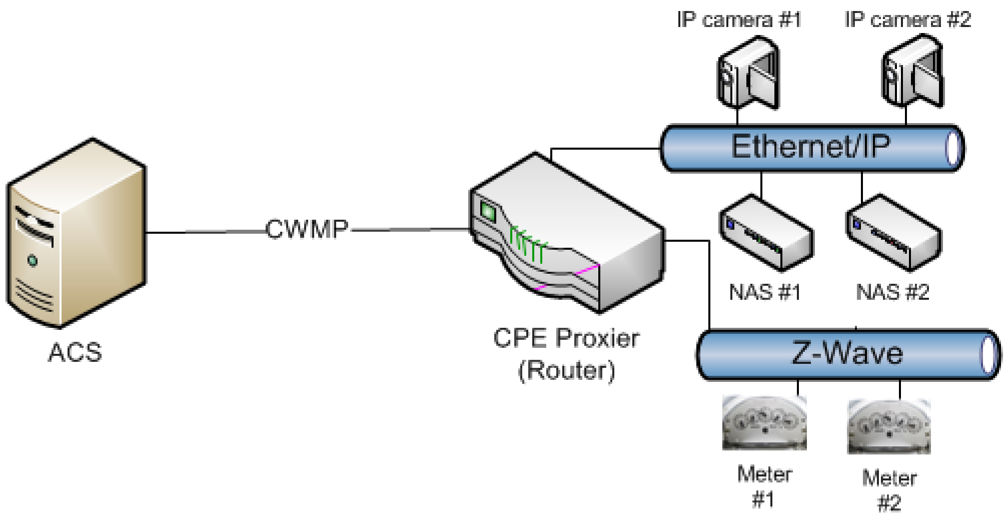


Figure 24 – Router supporting 6 Proxied Devices

Data Models; The ACS sees 3 Devices, a Router, NAS #1 and NAS #2:

<u>Router</u>

Device.DeviceInfo (DeviceInfo for the Router)
Device.DeviceInfo.SupportedDataModel.1.URL = RouterDevice URL
Device.DeviceInfo.SupportedDataModel.2.URL = IPCamera URL
Device.DeviceInfo.SupportedDataModel.3.URL = Meter URL
Device.ManagementServer.VirtualDevice.1.SerialNumber = (NAS #1 Serial Number)
Device.ManagementServer.VirtualDevice.1.ManufacturerOUI = (NAS #1 Manufacturer OUI)
Device.ManagementServer.VirtualDevice.1.ProductClass = (NAS #1 Product Class)
Device.ManagementServer.VirtualDevice.1.Host = (NAS #1 Host table entry)
Device.ManagementServer.VirtualDevice.1.ProxyProtocol = UPnP-DM
Device.ManagementServer.VirtualDevice.2.SerialNumber = (NAS #2 Serial Number)
Device.ManagementServer.VirtualDevice.2.ManufacturerOUI = (NAS #2 Manufacturer OUI)
Device.ManagementServer.VirtualDevice.2.ProductClass = (NAS #2 Product Class
Device.ManagementServer.VirtualDevice.2.Host = (NAS #2 Host table entry)
Device.ManagementServer.VirtualDevice.2.ProxyProtocol = UPnP-DM
Device.ManagementServer.ConnectionRequestURL = http://<IP>:8080/connreq
Device.ManagementServer.EmbeddedDevice.1.ControllerID = IP1
Device.ManagementServer.EmbeddedDevice.1.ProxiedDeviceID = 1
Device.ManagementServer.EmbeddedDevice.1.Reference = Device.Services.IPCamera.1
Device.ManagementServer.EmbeddedDevice.1.SupportedDataModel=Device.DeviceInfo.SupportedDataModel.2

Device.ManagementServer.EmbeddedDevice.1.Host = (IP Camera #1 in Host table)
Device.ManagementServer.EmbeddedDevice.1.ProxyProtocol = X_00256D_CamP
Device.ManagementServer.EmbeddedDevice.1.LastSyncTime = (last time the device was synced)
Device.ManagementServer.EmbeddedDevice.1.CommandProcessed = (state of last command)
Device.ManagementServer.EmbeddedDevice.2.ControllerID = IP1
Device.ManagementServer.EmbeddedDevice.2.ProxiedDeviceID = 2
Device.ManagementServer.EmbeddedDevice.2.Reference = Device.Services.IPCamera.2
Device.ManagementServer.EmbeddedDevice.2.SupportedDataModel=Device.DeviceInfo.SupportedDataModel.2
Device.ManagementServer.EmbeddedDevice.2.Host = (IP Camera #2 in Host table)
Device.ManagementServer.EmbeddedDevice.2.ProxyProtocol = X_00256D_CamP
Device.ManagementServer.EmbeddedDevice.2.LastSyncTime = (last time the device was synced)
Device.ManagementServer.EmbeddedDevice.2.CommandProcessed = (state of last command)
Device.ManagementServer.EmbeddedDevice.3.ControllerID = ZW1
Device.ManagementServer.EmbeddedDevice.3.ProxiedDeviceID = 1
Device.ManagementServer.EmbeddedDevice.3.Reference = Device.Meter.1
Device.ManagementServer.EmbeddedDevice.3.SupportedDataModel=Device.DeviceInfo.SupportedDataModel.3
Device.ManagementServer.EmbeddedDevice.3.Host = (Meter #1 in Host table)
Device.ManagementServer.EmbeddedDevice.3.ProxyProtocol = Z-Wave
Device.ManagementServer.EmbeddedDevice.3.LastSyncTime = (last time the device was synced)
Device.ManagementServer.EmbeddedDevice.3.CommandProcessed = (state of last command)
Device.ManagementServer.EmbeddedDevice.4.ControllerID = ZW1
Device.ManagementServer.EmbeddedDevice.4.ProxiedDeviceID = 2
Device.ManagementServer.EmbeddedDevice.4.Reference = Device.Meter.2
Device.ManagementServer.EmbeddedDevice.4.SupportedDataModel=Device.DeviceInfo.SupportedDataModel.3
Device.ManagementServer.EmbeddedDevice.4.Host = (Meter #2 in Host table)
Device.ManagementServer.EmbeddedDevice.4.ProxyProtocol = Z-Wave
Device.ManagementServer.EmbeddedDevice.4.LastSyncTime = (last time the device was synced)
Device.ManagementServer.EmbeddedDevice.4.CommandProcessed = (state of last command)
Device.Services.IPCamera.1
Device.Services.IPCamera.2
Device.Meter.1
Device.Meter.2

NAS #1

Device.DeviceInfo (DeviceInfo for the NAS #1)
Device.DeviceInfo.SupportedDataModel.1.URL = Device URL
Device.DeviceInfo.SupportedDataModel.2.URL = StorageService URL
Device.ManagementServer.ConnectionRequestURL = http://<IP>:8080/connreq-nas1
Device.ProxierInfo.SerialNumber = (Router Serial Number)
Device.ProxierInfo.ManufacturerOUI = (Router Manufacturer OUI)
Device.ProxierInfo.ProductClass = (Router Product Class)
Device.ProxierInfo.ProxyProtocol = UPnP-DM
Device.Services.StorageService.1

NAS #2

Device.DeviceInfo (DeviceInfo for the NAS #2)
Device.DeviceInfo.SupportedDataModel.1.URL = Device URL
Device.DeviceInfo.SupportedDataModel.2.URL = StorageService URL
Device.ManagementServer.ConnectionRequestURL = http://<IP>:8080/connreq-nas2
Device.ProxierInfo.SerialNumber = (Router Serial Number)
Device.ProxierInfo.ManufacturerOUI = (Router Manufacturer OUI)
Device.ProxierInfo.ProductClass = (Router Product Class)
Device.ProxierInfo.ProxyProtocol = UPnP-DM
Device.Services.StorageService.1

Appendix II. Alias-Based Addressing Mechanism – Theory of Operations

II.1 Introduction

This Appendix describes extensions to CPE WAN Management Protocol to allow for the OPTIONAL CPE Alias-Based Addressing Mechanism (defined in Section 3.6.1) for Multi-Instance Objects. This mechanism provides:

- Uniform Object identification across devices.
- Direct Object addressing using uniform identifiers.
- Ability to configure CPEs with less interrogation, including Object instance auto-creation.

The Alias-Based Addressing Mechanism provides an optional alternative to exclusively using the Instance Number based addressing. A CPE which supports the Alias-Based Addressing Mechanism can indicate this capability to the ACS, which can leverage it.

The Alias-Based Addressing Mechanism is provided to improve the end-to-end system scalability and robustness by allowing the ACS more direct control over how Objects are referenced during configuration or other management activities.

II.2 Multi-Instance Objects Definition

Multi-Instance Objects are designated in TR-069 Data Model documents with the "{i}" moniker. For example, TR-181 Issue 2 [35] defines an Object "Device.IP.Interface.{i}." and two of its Parameter names are: Name and Status.

Two IP Interface Objects and their Parameters, using Instance Numbers, might look as follows:

```
Device.IP.Interface.5.Name = "eth0"
Device.IP.Interface.5.Status = "Disabled"
Device.IP.Interface.30.Name = "fw0"
Device.IP.Interface.30.Status = "Disabled"
```

In the above example, the CPE has two Objects whose CPE assigned Instance Numbers are 5 and 30.

The Alias-Based Addressing Mechanism offers an alternative to use text of the ACS's choosing in place of Instance Numbers.

The same IP Interface Objects' Parameters, using Instance Aliases, might look as follows:

```
Device.IP.Interface.[wan].Name = "eth0"
Device.IP.Interface.[wan].Status = "Disabled"
Device.IP.Interface.[vpn].Name = "fw0"
Device.IP.Interface.[vpn].Status = "Disabled"
```

In above example, the CPE has two Objects whose ACS-assigned Instance Aliases are wan and vpn.

II.3 Instance Alias as a Data Model Parameter

In addition of its use within a Path Name to identify Object instances, an Alias is also a non-functional unique key Parameter. Therefore, an Alias Parameter can be handled as an ordinary TR-069 Parameter. For example, the following Path Name would return the value of Instance Alias Parameter if used with GetParameterValues RPC:

```
Device.IP.Interface.5.Alias
```

The Instance Alias can also be modified as a unique key on the referenced Object instance using a SetParameterValues.

For example using the Number-Based Addressing:

```
Device.IP.Interface.5.Alias = "lan"
```

or using the Alias-Based Addressing:

```
Device.IP.Interface.[wan].Alias = "lan"
```

II.4 Multi-Instance Object Creation

Object instances may come into being by one of the following ways:

- They might exist in CPE firmware, based on factory default configuration.
- If writeable they might be created on request of the ACS.
- They might be created by the CPE when it performs local functions such as configuration via local web UI, DHCP client discovery, Wi-Fi client discovery, alarm records, etc.

When the ACS needs to create a new Object instance, the CPE provides one via AddObject RPC. The CPE assigns a unique unpredictable Instance Number and subsequently the ACS has to refer to the created Object instance using the Instance Number assigned by the CPE.

With the Alias-Based Addressing Mechanism, the ACS can choose the Instance Alias that will be assigned to the Object instance at the time of its creation via SetParameterValues or AddObject RPC's. The ACS can then use the Instance Alias instead of the Instance Number to subsequently address the created Object instance.

With the Alias-Based Addressing Mechanism, the CPE is responsible for providing a unique Instance Alias in cases where it is not provided upon creation.

When a CPE creates a unique random Instance Alias, unlike the Instance Number the ACS can choose to modify the Instance Alias.

The Alias-Based Addressing Mechanism allows an ACS to choose a uniform naming mechanism for a particular Object to be used across multiple CPE devices such that they may be accessed using the same Instance Alias.

II.5 AddObject RPC Extension

The Alias-Based Addressing support extends the AddObject RPC to allow assigning an Instance Alias to the new Object instance during its creation using the Instance Alias notation (enclosed between square brackets) following the Path Name.

For example, if an AddObject RPC includes the Path Name Device.IP.Interface.1.IPv6-Address.[*ACS Assigned Instance Alias*]., then the CPE would perform the following:

- Create a new Object instance identified by Device.IP Interface.1.IPv6Addr-ess.*{CPE Assigned Instance Number}* and Device.IP.Interface.1.IPv6Addr-ess.*[ACS Assigned Instance Alias]*.
- The Parameter Device.IP.Interface.1.IPv6Address.*{CPE Assigned Instance Number}*.Alias is set to "*ACS Assigned Instance Alias*".
- The *{CPE Assigned Instance Number}* is returned to the ACS.

Subsequently the created Object instance and its Parameters are addressable as follows:

- By its Instance Number: Device.IP.Interface.1.IPv6Address.*{CPE Assigned Instance Number}*.*
- Or by its Instance Alias: Device.IP.Interface.1.IPv6Address.[*ACS Assigned Instance Alias*].*

II.6 Auto-Creation of Object Instances

If the ACS wants to modify a Parameter on an Object instance but does not know if the Object instance exists, it needs to query the CPE for the Object instance. If the Object instance does not exist, the ACS creates the Object instance with an AddObject RPC and then modifies the Parameter via a SetParameterValue command.

With the Alias-Based Addressing Mechanism enabled, and the Auto-Create Instance Mechanism enabled via the *ManagementServer.AutoCreateInstances* Parameter (described in Section A.3.2.1), the ACS can call a SetParameterValue for a Parameter in the Object instance using an Instance Alias in the Parameter Path Name. If the Object instance matching the Instance Alias does not exist, it will be created via the Auto-Create Instance Mechanism and the Parameter value is set.

The Alias-Based Addressing Mechanism saves time and resources both on the ACS and CPE, by avoiding the query of the CPE, as well as the additional round trip for each called RPC.

For example, if the Path Name Device.IP.Interface.1.IPv6Address.[*ACS Assigned Instance Alias*].IPAddress is used with SetParameterValues RPC to set the value "X", these actions would result:

- If does not already exist a new Object instance identified by Device.IP.-Interface.1.IPv6Address.*{CPE Assigned Instance Number}* or Device.IP.-Interface.1.IPv6Address.*[ACS Assigned Instance Alias]* is created.
- The Parameter Device.IP.Interface.1.IPv6Address.*{CPE Assigned Instance Number}*.Alias is set to "*ACS Assigned Instance Alias*",
- The Parameter Device.IP.Interface.1.IPv6Address.*{CPE Assigned Instance Number}*.IPAddress is set to "*X*".

II.7 Support for Alias-Based Addressing Mechanism

The CPE informs the ACS of its support for the Alias-Based Addressing Mechanism by using the ManagementServer.AliasBasedAddressing Parameter (set to true) within the Inform RPC.

If the ManagementServer.AliasBasedAddressing Parameter is absent in the Inform Parameters or its value set to false, The CPE will not provide the Alias-Based Addressing Mechanism.

If the ACS does not support the Alias-Based Addressing Mechanism it ignores the ManagementServer.AliasBasedAddressing received Parameter.

If the ACS supports the Alias-Based Addressing Mechanism it can use the Alias-Based Addressing associated Parameters only if the CPE has previously advertised support for this by providing the Device.ManagementServer.AliasBasedAddressing Parameter in the Inform.

Appendix III. XMPP Connection – Theory of Operations

III.1 Introduction

This Appendix describes a Theory of Operations around the XMPP.Connection table defined in tr-157-1-8.xml [41], which can be used by a CPE to communicate with an XMPP Server. One such use of this XMPP Connection Object and this type of communications is to facilitate the XMPP Connection Request mechanism as defined in Annex K. These XMPP Connection Objects can also be used for other communications that are yet to be defined by the Broadband Forum, but might be at some time in the future (or maybe for a purely proprietary means of communicating between the CPE and some entity connected to an XMPP Server).

III.2 XMPP Summary

This section provides a brief overview of the XMPP protocol for the purposes of assisting the reader in the remaining sections of this Appendix. The reader should be familiar with RFC 6120 [40], which is the authoritative source of this information.

XMPP is a communications protocol that permits bi-directional exchange of messages between entities using globally addressable clients and servers. In order to exchange messages between entities, the entities connect to a server as a client (called a “client-to-server” connection or more commonly a “c2s” connection). The clients may connect to the same or different servers. In the case where clients connect to multiple servers, the servers to which the client(s) connect are themselves connected (called a “server-to-server” connection or more commonly a “s2s” conection). In this architecture the CPE is a client to an XMPP server, and the ACS can also be a client to an XMPP Server (for the purpose of sending XMPP Connection Requests as defined in Annex K). Figure 25 in Appendix III.6 depicts an example deployment scenario where the ACS and CPE connect to different XMPP servers.

III.3 XMPP Identities

In order for entities to communicate using XMPP the entities are assigned globally unique addresses as defined in Section 2.1/RFC 6120 [40] (also known as a Jabber ID). In the context of this Appendix, the Jabber ID of interest is the one assigned to the CPE, by the first-hop XMPP Server, and further described in the following sub-section.

III.3.1 **CPE Considerations**

The Jabber ID is comprised of three parts: local-part, domain-part, and resource-part. These three parts are combined together in the following format: "local-part@domain-part/resource-part". A CPE will attempt to use the following Parameters from within the XMPP.Connection Object for the associated parts of the Jabber ID: the Username is the local-part, the Domain is the domain-part, and the Resource is the resource-part. The first-hop XMPP Server is ultimately responsible for determining the Jabber ID and informing the XMPP Client (the CPE) of what that Jabber ID is, thus there is also a JabberID Parameter within the XMPP.Connection Object that maintains the full Jabber ID that is dictated by the first-hop XMPP Server.

Typically a CPE's Jabber ID would use the CWMP unique triplet of OUI-ProductClass-SerialNumber as the local-part as that is a common unique identifier used within CWMP. An example of a Jabber ID for a CPE would be:

"OUI-ProductCode-SerialNumber@tr069.example.com/XMPPConn1"

where the:

- OUI-ProductCode-SerialNumber : represents the Username (local-part)
- tr069.example.com : represents the Domain (domain-part)
- XMPPConn1 : represents the Resource (resource-part)

III.4 **Establishing an XMPP Connection**

In order to exchange messages between XMPP clients, the client needs to establish an XMPP Connection to the configured XMPP Server. The XMPP Client establishes this connection by establishing two XML Streams as detailed in the following process (from the perspective of the CPE):

- Discover the address of the XMPP Server
- Open a TCP Connection to the XMPP Server
- Open an XML Stream to the XMPP Server
- Accept an XML Stream from the XMPP Server
- Authenticate the XMPP channel using SASL credentials
- Bind the resource to determine the full Jabber ID of the XMPP Client

Once the CPE establishes the XML Streams, the CPE can then exchanges messages with other XMPP Clients using XML Stanzas. Annex K defines how an ACS (acting as an XMPP Client) can issue an XMPP Connection Request via an XML Stanza.

III.4.1 **Determining the XMPP Server Address and Port**

In order for an XMPP Client to connect to an XMPP Server, the client must know the IP address and port of the server. The XMPP.Connection Object provides multiple mechanisms to determine the XMPP Server to which the XMPP Client connects:

- *DNS-SRV* - Use the connection algorithm as specified in Section 3.2/RFC 6120 [40] where the value of the XMPP.Connection.{i}.Domain parameter is used to look up the server address and port to use. This is the default connection algorithm.
- *DNS* - Use the fallback connection mechanism as specified in Section 3.2.2/RFC 6120 [40], where the value of the XMPP.Connection.{i}.Domain parameter is used to look up the server address, and the port is the "xmpp-client" port (5222).
- *ServerTable* - Use the instances of the Server table (a table on the XMPP.Connection instance) based on the value of a Server instance's Priority and Weight parameter values to determine the appropriate server address and port to use.
- *WebSocket* - Use the connection algorithm as specified in Section 4/RFC 7395 [58] where the value of the Domain parameter is used to assemble the URL of a host-metadata document which should be fetched to discover the service URL for the WebSocket over which to establish the XMPP connection.

These options are provisioned in the XMPP Connection using the ServerConnect-Algorithm Parameter within the XMPP.Connection Object.

III.4.2 **XML Streams**

XML Streams are unidirectional, meaning that there needs to be a stream from the XMPP Client to the XMPP Server and another stream from the XMPP Server to the XMPP Client as the messages only flow in one direction on each stream. The concept of directionality only applies to XML Stanzas that are sent after the XML Streams have been established (meaning that it doesn't apply to TLS negotiation or SASL authentication). These two XML Streams can occupy one or two TCP connections, but according to Section 4.5/RFC 6120 [40], the most common approach for "client-to-server" connections is to use a single TCP connection for both XML Streams where the security context negotiated for the first stream is applied to the second stream.

III.4.3 **XMPP Connection Encryption**

During the process of establishing an XMPP connection, if the XMPP Server is configured such that TLS is "mandatory-to-negotiate", then the XMPP connection needs to be either upgraded to a TLS encrypted XMPP channel as defined in Section 5/RFC 6120 [40] or connected over a WebSocket Secure connection as explained in Section 3.9/RFC 7395 [58]..

III.4.4 **XMPP Channel Authentication**

Once the XMPP connection is encrypted, the XMPP Connection is authenticated using the Simple Authentication and Security Layer (SASL) protocol as defined in Section 6/RFC 6120 [40]. The Username and Password Parameters of the XMPP.Connection Object are used as the credentials for the SASL authentication procedure.

III.4.5 Reconnecting to the XMPP Server

There are some maintenance and unexpected scenarios where the XMPP Server, to which an XMPP Client has connected, goes offline. In this situation the XMPP Client needs to reconnect with the XMPP Server in order to maintain uninterrupted service. Section 3.3/RFC 6120 [40] defines a set of reconnection requirements for an XMPP Client. The following recommendations are also to be followed by CPE that establish an XMPP Connection.

If a CPE loses its connection to the XMPPP Server, it needs to reconnect using the set of XMPP Servers configured in the ServerConnectAlgorithm Parameter associated with the XMPP.Connection instance that needs to be reconnected.

In situations where there are long outages, many CPE that implement Section 3.3/RFC 6120 [40] will abort retry requests for a specific server after some number of attempts. In order to provide more control, the recommendation here is to utilize the ServerConnectAttempts, ServerRetryInitialInterval, ServerRetryIntervalMultiplier, and ServerRetryMaxInterval Parameters associated with the XMPP.Connection instance that needs to be reconnected.

III.5 Maintaining an XMPP Connection

The XMPP Client connection to an XMPP Server is considered to be a long lived TCP session. As such, the XMPP client needs to periodically send Keep-Alive messages to the server in order to maintain the TCP session. The preferred mechanism is to utilize the “whitespace keepalive” mechanism defined in Section 4.6.1/RFC 6120 [40] where an XMPP Client sends a single space character between XML Stanzas.

The KeepAliveInterval Parameter within the XMPP.Connection Object allows an ACS to configure the keepalive frequency in the following ways:

- Value = -1 : the whitespace keepalive mechanism is implementation specific and determined by the CPE (default option)
- Value = 0 : the whitespace keepalive mechanism is disabled (not recommended)
- Value > 0 : the whitespace keepalive messages should be sent every KeepAliveInterval seconds as configured by the ACS

III.6 Deployment Considerations

Firewalls are typically classified as one of three different types: Packet Filter Firewall, Application Layer Firewall, and Stateful Firewall. A Packet Filter Firewall simply filters each packet based on the information contained within the packet itself, meaning that it does not pay attention to whether the packet is part of an existing stream of traffic. An Application Layer Firewall operates on all 7 layers of the stack, instead of the first 3 layers like the Packet Filter Firewall, so it has the capability of not only filtering packets but also filtering information based on content. A Stateful Firewall does everything that an Application Layer Firewall does but also maintains records for every connection passing through the firewall and can thus perform stateful packet inspection of that information.

This means that if a Stateful Firewall is being utilized to separate the CPE from the ACS and XMPP Servers then there could be load/scale issue for it to maintain all of the open connections between the XMPP Clients on the CPE and the XMPP Server cluster. A work around to this concern would be to place an Application Layer Firewall in front of the Stateful Firewall and have the main CPE-bearing XMPP cluster resides between the Application Layer Firewall and the Stateful Firewall. The XMPP Client that the ACS utilizes could still reside behind the Stateful Firewall and utilize a small cluster of XMPP Servers that also reside behind the Stateful Firewall, meaning that there would be a single server-to-server connection between the 2 sets of XMPP Server clusters (see below for a visual depiction of this configuration).

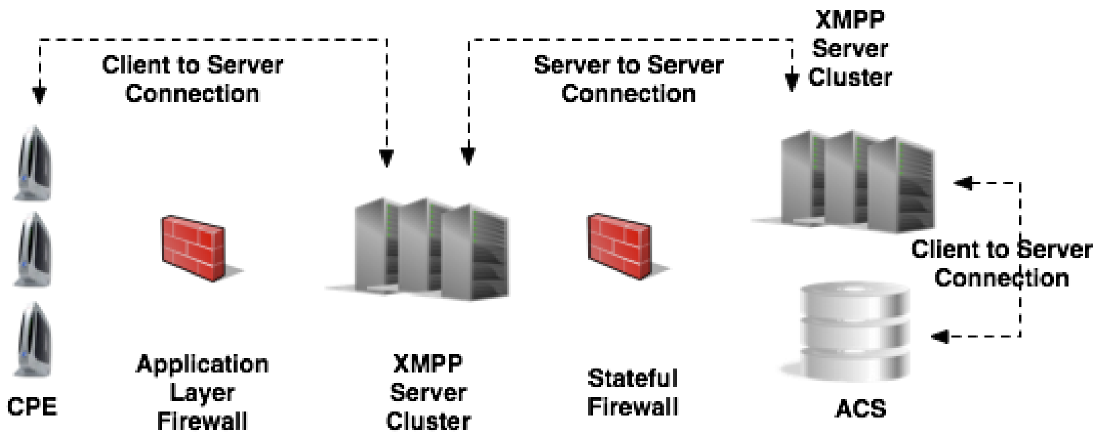

Client to Server Connection
Server to Server Connection
XMPP Server Cluster
Client to Server Connection
CPE
Application Layer Firewall
XMPP Server Cluster
Stateful Firewall
ACS

Figure 25 – XMPP Deployment Option – Stateful Firewall

Appendix IV. UPnP IGD and HTTP Connection Requests

IV.1 Introduction

This section describes a mechanism allowing a CPE that is not directly addressable by its ACS to make standard Connection Requests possible, by taking advantage of the UPnP Internet Gateway Device framework. This mechanism is an alternative to the XMPP-based solution defined in Annex K, but this mechanism is only applicable if the gateway, behind which the CPE resides, supports UPnP IGD [42] and [43] with the necessary services and allows the CPE to use them. Two different kinds of actions may be required from the CPE: managing port mappings and/or creating a pinhole in an IPv6 firewall.

IV.2 NAT Port Mapping Theory of Operation

The UPnP Internet Gateway Device framework allows management of Port Mappings in a NAT-enabled gateway, which can be used to allow Connection Requests to reach a CPE from the WAN. This is only possible if the gateway implements the WANIPConnection:2 service [45] or the WANPPPConnection:1 service [44], depending on the WAN access protocol in use. The CPE also needs to embed a UPnP Control Point supporting the UPnP WANIPConnection:2 and WANPPPConnection:1 services. The general procedure has the CPE discovering the gateway's WAN interface settings and IP address, which allows it to open a port mapping from an external port towards the port the CPE wants to use for receiving Connections Requests, and concludes with the CPE setting its ConnectionRequestURL value based on information retrieved and configured through UPnP IGD.

The UPnP IGD framework and methods are also used to manage changes of Connection Request URL related parameters when an IP address (CPE, gateway, or ACS) is modified or when the CPE or gateway is shutting down.

The discovery phase is independent from the WAN protocol and is described in Section IV.2.1, but the other operations differ: the PPP case (WANPPPConnection:1 Service) is described in Section IV.2.2 and the IP case (WANIPConnection:2 Service) in Section IV.2.3.

IV.2.1 Discovery and External IP Address Retrieval

The CPE first discovers the UPnP IGD Gateway on the local network by sending an SSDP M-SEARCH request.

Once the UPnP IGD Gateway detects that it is being searched for, it sends back a unicast message to the CPE. The information contained in this response allows the CPE to retrieve the description of the gateway and all necessary information concerning the services that it hosts.

The CPE checks whether the gateway supports a WANPPPConnection:1 service or a WANIPConnection:2 service. If the gateway supports neither the WANPPPConnection:1 service nor the WANIPConnection:2 service, or the gateway supports multiple WAN interfaces, the CPE needs to use an alternate solution.

When the UPnP IGD Gateway is found, the CPE uses the subscription mechanism provided by UPnP in order to be notified of changes. Then, right after the subscription, the CPE LAN Device gets the WAN IP address via a Gateway notification. By using the UPnP notification/subscription mechanism, the CPE does not have to use the GetExternalIPAddress method. The CPE MUST continue to monitor for advertisements from the UPnP IGD Gateway so as not to miss reboots.

IV.2.2 **Procedures for WANPPPConnection:1**

Before creating a new port mapping entry, the CPE needs to check whether the port mapping already exists. This is done by invoking the GetSpecificPortMappingEntry UPnP action on the gateway. The input arguments for this UPnP action are as follows:

Input argument	Recommended Contents
NewRemoteHost	The ACS IP address
NewExternalPort	The port on which Connection Request packets issued by the ACS will be received by the gateway
NewProtocol	The protocol used for the Connection Request

The result of the GetSpecificPortMappingEntry action provides the CPE with the knowledge of whether or not the port mapping entry exists.

If the port mapping entry already exists, the NewInternalClient value, returned by the Gateway, provides information regarding which device the packets are being forwarded to. If the existing port mapping entry is used by another client (NewInternalClient value is not the IP address of the CPE), then the CPE chooses a new external port and tries again. If the existing port mapping entry is for this device (NewInternalClient value is the IP address of the CPE), then the CPE uses the existing port mapping entry.

If the port mapping entry does not exist, the CPE attempts to create a new port mapping entry by invoking the AddPortMapping UPnP action. The input arguments for this UPnP action are as follows:

Input argument	Recommended Contents
NewRemoteHost	The ACS IP address (IP filter security is possible)
NewExternalPort	The port on which Connection Request packets issued by the ACS will be received by the gateway
NewProtocol	The protocol used for the Connection Request
NewInternalPort	The port that the CPE is listening to for Connection Requests sent by the ACS

Input argument	Recommended Contents
NewInternalClient	The IP address of the CPE
NewLeaseDuration	The lease duration of the port mapping entry. The CPE would typically use a small value (e.g., less than 5 minutes) to avoid port maping cleanup issues
PortMappingDescription	Human-readable description associated to this port mapping entry (optional)

If the action fails, an error response is returned to the CPE. When this happens, the following actions can be taken (based on the error that was returned):

- "Action failed": the UPnP action failed and the CPE needs to use an alternate solution
- "Conflict in mapping entry": the CPE needs to restart the procedure with a different NewExternalPort value (e.g. former NewExternalPort +1)

If the port mapping entry has been successfully created or identified, the CPE sets the ConnectionRequestURL value (http://host:port/path) of its Data Model where:

- The “host” value is the discovered WAN IP address of the Gateway (ExternalIPAddress)
- The “port” value is the NewExternalPort either from the GetSpecificPortMappingEntry UPnP action (if the port mapping entry already existed and was associated to this device) or from the AddPortMapping UPnP action (if a new port mapping entry was created)
- The “path” value is the path the CPE wants to use

If the ConnectionRequestURL parameter is configured for Active notifications (which is typically default behaviour), the CPE issues an Inform RPC to the ACS with a “4 VALUE CHANGE” event code whenever the ConnectionRequestURL value is altered.

Once the port mapping entry has been established, the CPE is responsible for keeping the port mapping entry from expiring. This can be done by periodically invoking the AddPortMapping UPnP action, using the same parameter values as the existing port mapping entry.

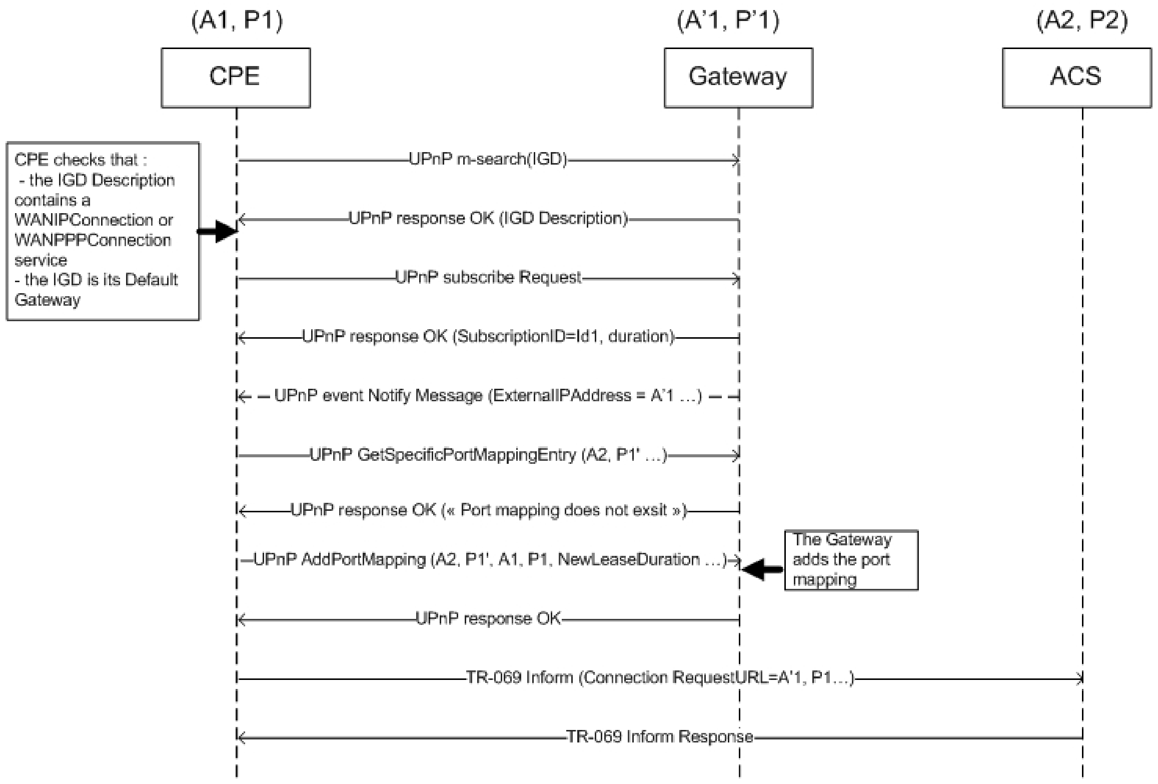

(A1, P1)
CPE
(A'1, P'1)
Gateway
(A2, P2)
ACS
CPE checks that :
- the IGD Description contains a WANIPConnection or WANPPPConnection service
- the IGD is its Default Gateway
UPnP m-search(IGD)
UPnP response OK (IGD Description)
UPnP subscribe Request
UPnP response OK (SubscriptionID=Id1, duration)
UPnP event Notify Message (ExternalIPAddress = A'1 …)
UPnP GetSpecificPortMappingEntry (A2, P1' …)
UPnP response OK (« Port mapping does not exsit »)
UPnP AddPortMapping (A2, P1', A1, P1, NewLeaseDuration …)
The Gateway adds the port mapping
UPnP response OK
TR-069 Inform (Connection RequestURL=A'1, P1…)
TR-069 Inform Response

Figure 26 – Message flows when the CPE LAN device starts up normally

IV.2.3 Procedures for WANIPConnection:2

The CPE attempts to create a port mapping entry by invoking the AddAnyPortMapping UPnP action. The input arguments for this UPnP action are as follows:

Input argument	Recommended Contents
NewRemoteHost	The ACS IP address (IP filter security is possible)
NewExternalPort	The port on which Connection Request packets issued by the ACS will be received by the gateway
NewProtocol	The protocol used for the Connection Request
NewInternalPort	The port that the CPE is listening to for Connection Requests sent by the ACS
NewInternalClient	The IP address of the CPE
NewLeaseDuration	The lease duration of the port mapping entry. The CPE would typically use a small value (e.g., less than 5 minutes) to avoid port mapping cleanup issues, and the gateway removes dynamic port mapping entries across resets or reboots
PortMappingDescription	Human-readable description associated to this port mapping entry (optional)

If the action fails, an error response is returned to the CPE. When this happens, the CPE needs to use an alternate solution.

If the gateway successfully creates the port mapping entry, a response containing a NewReservedPort argument is sent back to the CPE. The value of NewReservedPort might be different from the one proposed by the CPE, and this is why there is no need to check the existing port mapping entries before attempting to create a new one, as is done in the procedures defined in Section IV.2.2.

When the port mapping entry has been successfully created, the CPE sets the ConnectionRequestURL value (http://host:port/path) of its Data Model where:

The “host” value is the discovered WAN IP address of the Gateway (ExternalIPAddress)

The “port” value is the NewReservedPort from the AddAnyPortMapping UPnP action

The “path” value is the path the CPE wants to use

If the ConnectionRequestURL parameter is configured for Active notifications (which is typically default behaviour), the CPE issues an Inform RPC to the ACS with a “4 VALUE CHANGE” event code whenever the ConnectionRequestURL value is altered.

Once the port mapping entry has been established, the CPE is responsible for keeping the port mapping entry from expiring. This can be done by periodically invoking the AddAnyPortMapping UPnP action, using the same parameter values as the existing port mapping entry (according to the UPnP specification a call to AddAnyPortMapping with the same Protocol, ExternalPort, RemoteHost, and InternalClient of an existing port mapping entry will simply refresh the lease duration instead of creating a new port mapping entry with a different reserved port).

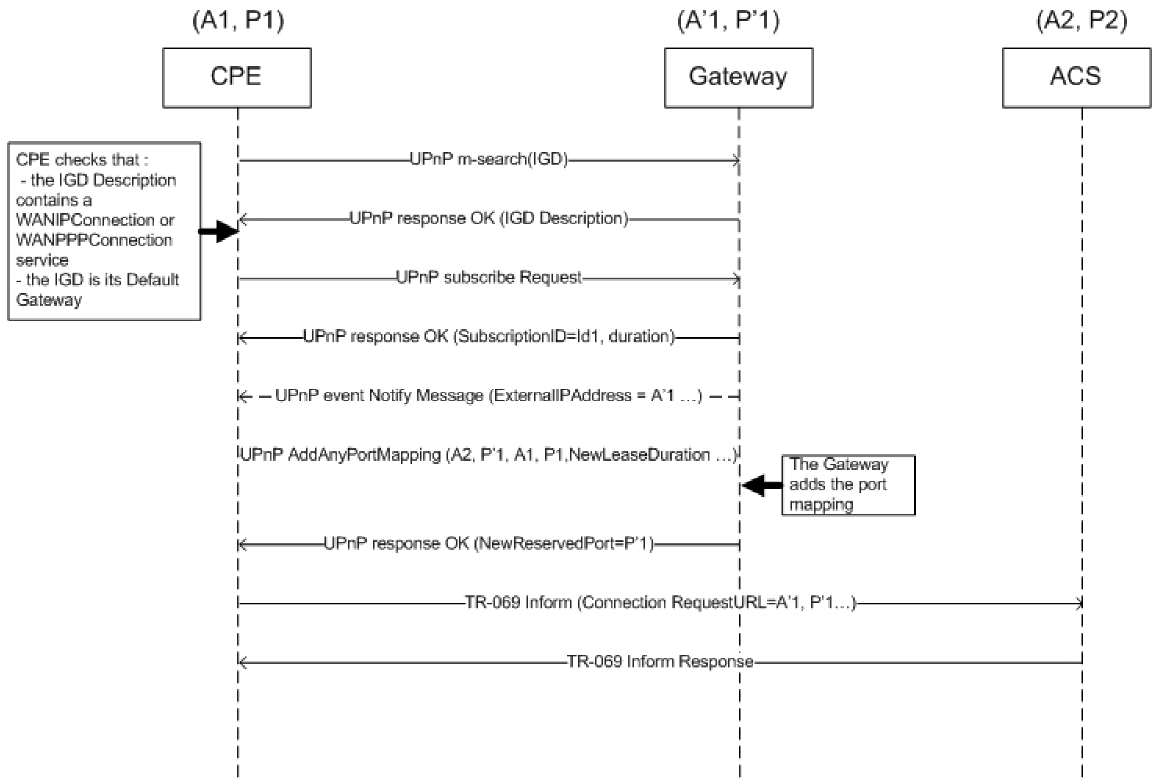

(A1, P1)
(A'1, P'1)
(A2, P2)
CPE
Gateway
ACS
CPE checks that :
- the IGD Description contains a WANIPConnection or WANPPPConnection service
- the IGD is its Default Gateway
UPnP m-search(IGD)
UPnP response OK (IGD Description)
UPnP subscribe Request
UPnP response OK (SubscriptionID=Id1, duration)
UPnP event Notify Message (ExternalIPAddress = A'1 ...)
UPnP AddAnyPortMapping (A2, P'1, A1, P1,NewLeaseDuration ...)
The Gateway adds the port mapping
UPnP response OK (NewReservedPort=P'1)
TR-069 Inform (Connection RequestURL=A'1, P'1...)
TR-069 Inform Response

Figure 27 – Message flows when the CPE LAN device starts up normally

IV.2.4 **Handling Changes**

When the CPE reboots or its IP address changes, it applies the above procedures again to restore the appropriate mapping and communicate the ConnectionRequestURL change to the ACS.

When the gateway reboots or its IP address changes, the new address is notified to the CPE through its UPnP IGD subscription, at which point the CPE needs to apply the above procedures again to restore the appropriate mapping and communicate the ConnectionRequestURL change to the ACS.

IV.3 **IPv6 Pinhole Theory of Operation**

RFC 6092 [47] states that "*the general operating principle is that transport layer traffic is not forwarded into the interior network of a residential IPv6 gateway unless it has been solicited explicitly by interior transport endpoints.*" As such, Section LAN.PFWDv6.1 of TR-124 Issue 3 [48] requires that Connection Requests sent by the ACS to a CPE that resides behind an IPv6 firewall are dropped.

The UPnP Internet Gateway Device framework allows management of IPv6 Inbound Pinholes in the IPv6 Firewall of the gateway, which allow Connection Requests to reach the CPE from the WAN. This is only possible if the gateway and CPE implement the UPnP IGD WANIPv6FirewallControl:1 service [46]. The general procedure has the CPE discovering the gateway's status of the embedded firewall, which allows it to create and maintain an Inbound Pinhole in the IPv6 Firewall rules of the gateway for the purpose of relaying Connection Request messages to the CPE.

The UPnP IGD framework and methods are also used to manage changes of Connection Request URL related parameters when an IP address (CPE, gateway, or ACS) is modified or when the CPE or gateway is shutting down.

IV.3.1 Discovery

The CPE first discovers the UPnP IGD Gateway on the local network by sending an SSDP M-SEARCH request.

Once the UPnP IGD Gateway detects that it is being searched for, it sends back a unicast message to the CPE. The information contained in this response allows the CPE to retrieve the description of the gateway and all necessary information concerning the services that it hosts.

The CPE checks whether the gateway has a WANIPv6FirewallControl:1 service. If the gateway doesn’t support the WANIPv6FirewallControl:1, the CPE needs to use an alternative solution.

IV.3.2 Procedures

The CPE retrieves the IPv6 firewall status and the ability to add pinholes by invoking the GetFirewallStatus UPnP action (there are no arguments for this UPnP action).

- If the firewall is disabled, the LAN device doesn't have to worry about the firewall blocking Connection Requests from the ACS
- If the firewall is enabled and the CPE is not allowed to create Inbound Pinholes, the CPE needs to use an alternative solution

The CPE opens a pinhole on the firewall of the gateway for the port on which the CPE is listening on for Connection Requests by invoking the AddPinhole UPnP action. The input arguments for this UPnP action are as follows:

Input argument	Recommended Contents
RemoteHost	The ACS IP address (IP filter security is possible)
RemotePort	The port on which Connection Request packets issued by the ACS will be received by the gateway
InternalClient	The IP address of the CPE
InternalPort	The port that the CPE is listening to for Connection Requests sent by the ACS
LeaseTime	The lease duration of the pinhole. The CPE would typically use a small value (e.g., less than 5 minutes) to avoid pinhole cleanup issues
Protocol	The protocol used for the Connection Request

The gateway might have sent back an error to the CPE as a result of calling the AddPinhole UPnP action. In this case, the CPE needs to use an alternative solution.

When the Inbound Pinhole has been successfully create, the CPE sets the ConnectionRequestURL value (http://host:port/path) of its Data Model where:

- The “host” value is the IPv6 address of the CPE
- The “port” value is the port opened in the Firewall
- The “path” value is the path the CPE wants to use

If the ConnectionRequestURL parameter is configured for Active notifications (which is typically default behaviour), the CPE issues an Inform RPC to the ACS with a “4 VALUE CHANGE” event code whenever the ConnectionRequestURL value is altered.

Once the Inbound Pinhole has been established, the CPE is responsible for keeping the pinhole from expiring. This can be done by periodically invoking the UpdatePinhole UPnP action. The input arguments for this UPnP action are as follows:

Input argument	Recommended Contents
UniqueID	The unique ID of the Inbound Pinhole returned by original AddPinhole UPnP action
NewLeaseTime	The lease duration of the Inbound Pinhole. The CPE would typically use a small value (ie, less than 5 minutes) to avoid pinhole cleanup issues

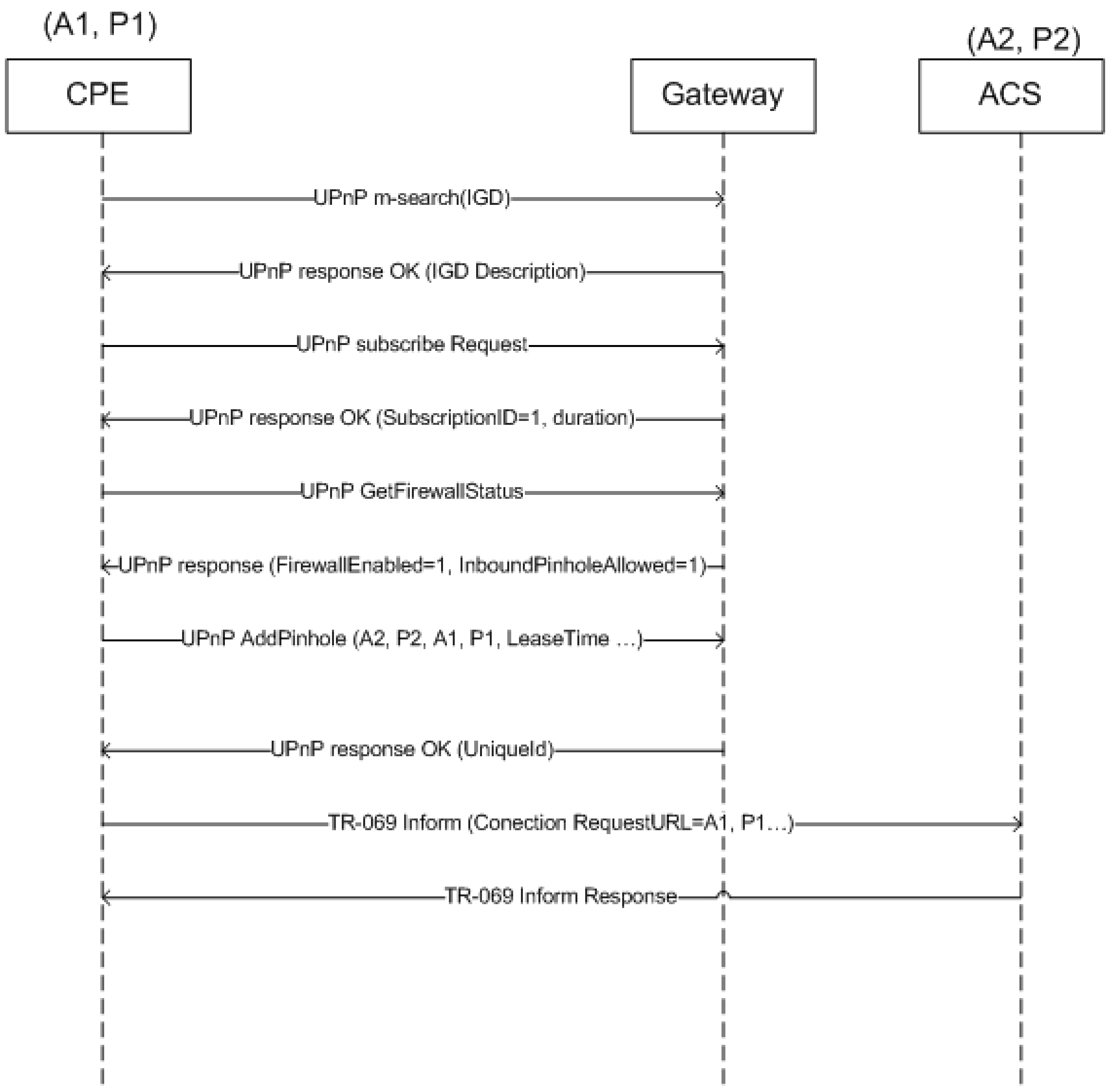

(A1, P1)
CPE
Gateway
(A2, P2)
ACS
UPnP m-search(IGD)
UPnP response OK (IGD Description)
UPnP subscribe Request
UPnP response OK (SubscriptionID=1, duration)
UPnP GetFirewallStatus
UPnP response (FirewallEnabled=1, InboundPinholeAllowed=1)
UPnP AddPinhole (A2, P2, A1, P1, LeaseTime …)
UPnP response OK (UniqueId)
TR-069 Inform (Conection RequestURL=A1, P1…)
TR-069 Inform Response

Figure 28 – Message flows when the CPE LAN device starts up normally

IV.3.3 Handling Changes

When the CPE reboots or its IP address changes, it applies the above procedures again to restore the appropriate mapping and communicate the ConnectionRequestURL change to the ACS.

When the gateway reboots or its IP address changes, the new address is notified to the CPE through its UPnP IGD subscription, at which point the CPE needs to apply the above procedures again to restore the appropriate pinhole and communicate the ConnectionRequestURL change to the ACS.

Appendix V. Multiple Firmware Images – Theory of Operation

V.1 Introduction

Many manufacturers are now starting to build and deploy devices that are able to support multiple firmware images (i.e. multiple firmware images can be installed on the CPE at the same time). There are at least a couple of advantages to this strategy:

1. Having multiple firmware images installed improves the robustness and stability of the device because, in all likelihood, one of the installed images will be stable and bootable. Should a device not be able to boot a newly installed firmware image, it could have the ability to attempt to boot from a different firmware image, thus allowing the device to come back online,
2. Support for multiple firmware images offers the ability for the service provider to have a new firmware downloaded (but not activated) to the device at any point during the day, then perhaps requiring only a call to SetParameterValues and Reboot at some later time (perhaps during a short maintenance window or when the device is idle) to cause the device to switch over to the new firmware. Along with reducing the impact on the subscriber, the ability to spread the download portion a firmware upgrade over a longer period of time (eg, the entire day or over several days) can help minimize the impact of the upgrade on the provider's network.

This Appendix discusses how to take advantage of a device's support of multiple firmware images.

V.2 Getting the firmware image onto the device

V.2.1 New firmware image file type

The original way to perform a firmware upgrade typically involves the following steps:

1. ACS issues a Download RPC to a device, specifying a file type of "1 Firmware Upgrade Image"
2. The device downloads the specified file

3. The device validates and installs the firmware image
4. The device boots the new firmware image. At this point, the file is considered to be "applied"
5. The device initiates a Session to the ACS
6. The device sends a TransferComplete RPC, thereby completing the firmware upgrade

The key point here is the definition of "applied." A firmware image, downloaded via file type "1 Firmware Upgrade Image," is "considered to be successfully applied only after the new software image is actually installed and operational." (see section A.3.2.8) For the use case where a service provider wishes to have an additional firmware image installed, but not running, the definition of "applied" is different than that defined for the "1 Firmware Upgrade Image" file type.

The definition of "applied" for the original firmware upgrade file type cannot be altered, so the multiple firmware image mechanism introduces a new file type: "6 Stored Firmware Image". This new file type is to be treated exactly the same as the original firmware upgrade file type, with the only difference being that files of this type are considered to be "applied" when they are downloaded, validated, and installed only. That is, the new firmware image does not actually have to be running in order for the device to consider the download to be completed.

V.2.2 **Where does the firmware image go?**

When a device receives a Download request with the "6 Stored Firmware Image" file type, the device needs to download the image and install it into a non-active firmware image location. A firmware image location is non-active if it is not the location containing the currently running firmware image.

If a device has support for only two firmware images, one firmware image location will be active (i.e. it holds the firmware that is currently in operation) and the other will be inactive. For such devices, determining which location to install a new firmware image is a simple process. For devices that support more than two firmware images, which inactive location to use for the new image is beyond the scope of this Appendix.

V.3 **Using Multiple Firmware Images**

V.3.1 **Switching firmware images**

Once a device has multiple firmware images installed and available, an ACS can use the data model to query what images are on the device, which image is active, and configure which image the device should load the next time it boots.

To get a device to boot into the new firmware, the ACS needs to modify the "Device.DeviceInfo.BootFirmwareImage" parameter to point to the "Device.DeviceInfo.-FirmwareImage.{i}." object instance that contains the new firmware image. To make this determination, the ACS may need to check each FirmwareImage object to find the one with the desired firmware image (this discovery process would be useful for devices that support more than two firmware images).

After setting "Device.DeviceInfo.BootFirmwareImage" to the appropriate value, the ACS can then trigger the device to reboot (either by issuing a Reboot RPC to the device or by making use of the "Device.ManagementServer.ScheduleReboot" or "Device.-ManagementServer.DelayReboot" parameters). This should then cause the device to boot up into the specified firmware.

When attempting to get a device to switch to a different firmware image, it is recommended that the ACS pay attention to the value of the DeviceInfo.SoftwareVersion parameter when the device sends an Inform. If the value has not changed from previous Informs or is not the expected value, it could be an indication that the device had problems booting the target firmware image.

V.3.2 **Performing a delayed firmware upgrade**

As stated in the introduction, one of the benefits to having support for multiple firmware images on a device is that it provides an opportunity to push a firmware image to a device and then have the device switch to that image at a later time. This functionally allows a service provider to push a firmware image to a set of devices at any point during the day and then use a maintenance window to switch all of the target devices to the target firmware.

This ability is of value because normally the download of the firmware and the switch to the new image would both have to take place during the maintenance window. Bandwidth limitations may have an impact on the number of devices that can be performing the download at the same time. If this is the case, the number of devices that can be upgrading at the same time may be lower than desired, requiring multiple maintenance windows to complete the upgrade. However, support for multiple firmware images allows for the service provider to push firmware images over a longer period of time and then use a smaller maintenance window to tell the device to switch firmware images. This can result is shorter system-wide firmware upgrades.

V.3.3 **Recovering from a failed upgrade**

As stated in the introduction, one of the benefits of having multiple firmware images on a device is that if a device cannot boot into a target firmware image because of some problem with the image, the device could then try to boot one of the other firmware images.

When there are two images, the device would simply try booting the alternate image (which, ideally, holds the previous version of the firmware). If there are more than two images, the device could try booting from any of the other available images. Ideally, the device would keep track of and try to boot from the previously known working firmware (assuming that firmware is still installed on the device).

Should the device boot a firmware image other than that specified via the "Device.DeviceInfo.BootFirmwareImage" parameter, it is important that the device MUST NOT change the value of the "Device.DeviceInfo.BootFirmwareImage" parameter to point to the currently-running firmware image object. If the device was to change this parameter value, it could make troubleshooting problems with a firmware image switch more difficult.

As recommended in section IV.3.1, the ACS should be keeping track of the value of “Device.DeviceInfo.SoftwareVersion” parameter. If the version changes unexpectedly, it could be an indication that the device had problems booting a particular firmware image.

V.3.4 Backwards compatibility

While this Appendix defines a mechanism for installing additional firmware images onto a device that can be activated at a later time, it’s important to remember that this mechanism offers supplemental functionality. If a device receives a Download with file type “1 Firmware Upgrade Image”, it must perform the firmware upgrade as originally defined in A.3.2.8/A.4.1.8. In these circumstances, the device should install the new firmware image over top of the currently active firmware (ie, the same firmware location).

Appendix VI. Software Module Management

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

VI.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.

VI.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.

VI.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.

VI.3.1 **Deployment Units**

Below is the state machine diagram[37] for the lifecycle of DUs.

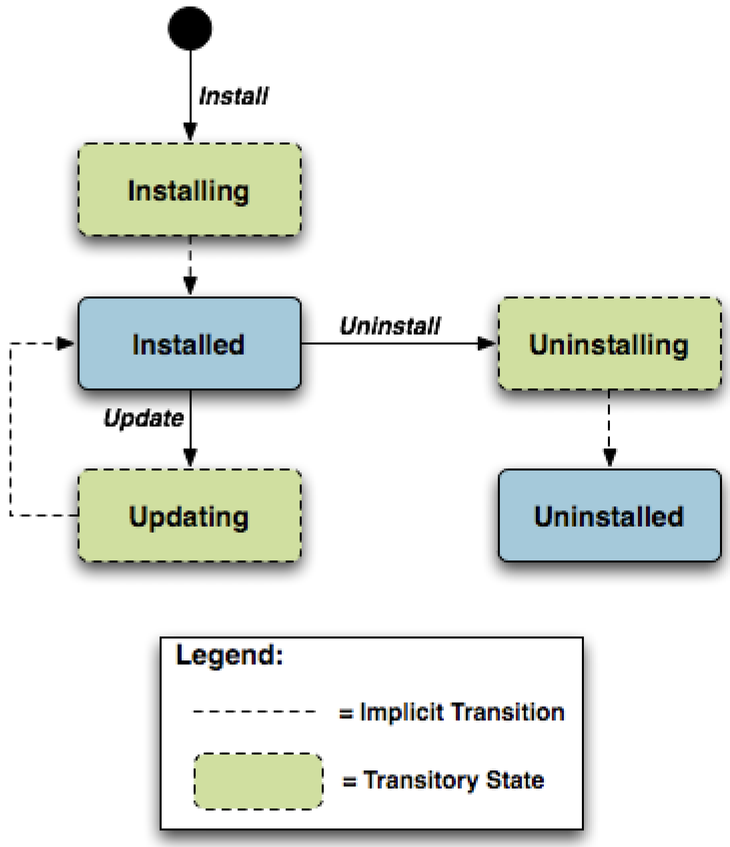


Figure 29 – Deployment Unit State Diagram

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. The explicit transitions are as follows:

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

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 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.. 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 Section VI.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.

VI.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 [37] Version 5 and TR-181i2a12 Annex C [35]. 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.

VI.3.2 **Execution Units**

Below is the state machine diagram[38] for the lifecycle of EUs.

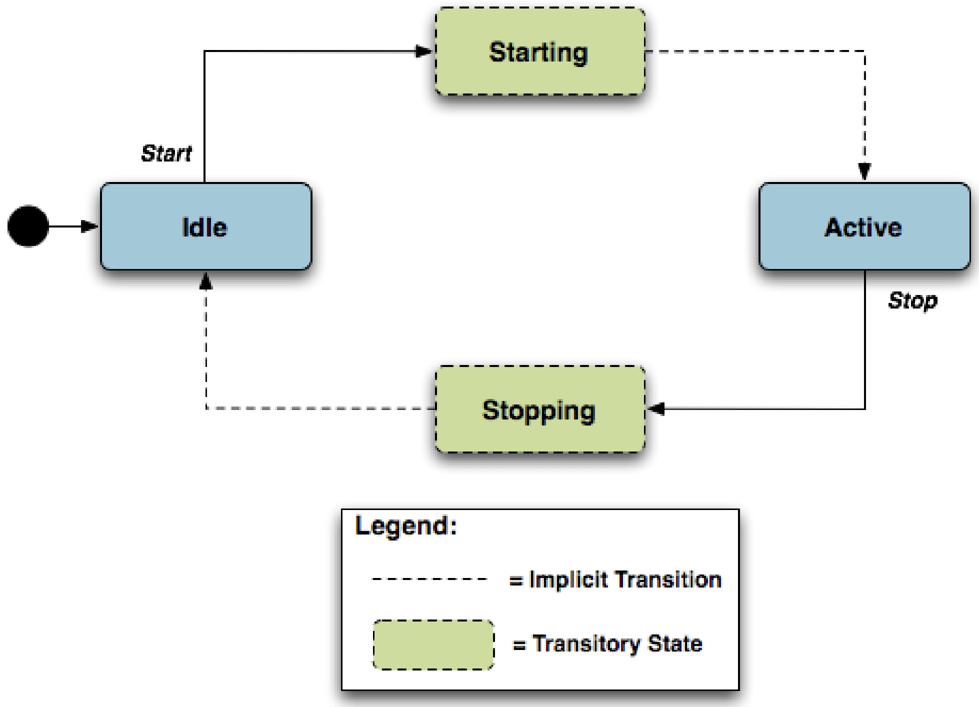


Figure 30 – Execution Unit State Diagram

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 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,

[38] 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.

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 [28]).
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 [27] or TR-181 Issue 2 [35]).
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 A.4.1.5 and Download A.3.2.8 methods, 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 A.4.1.5 method.

VI.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.

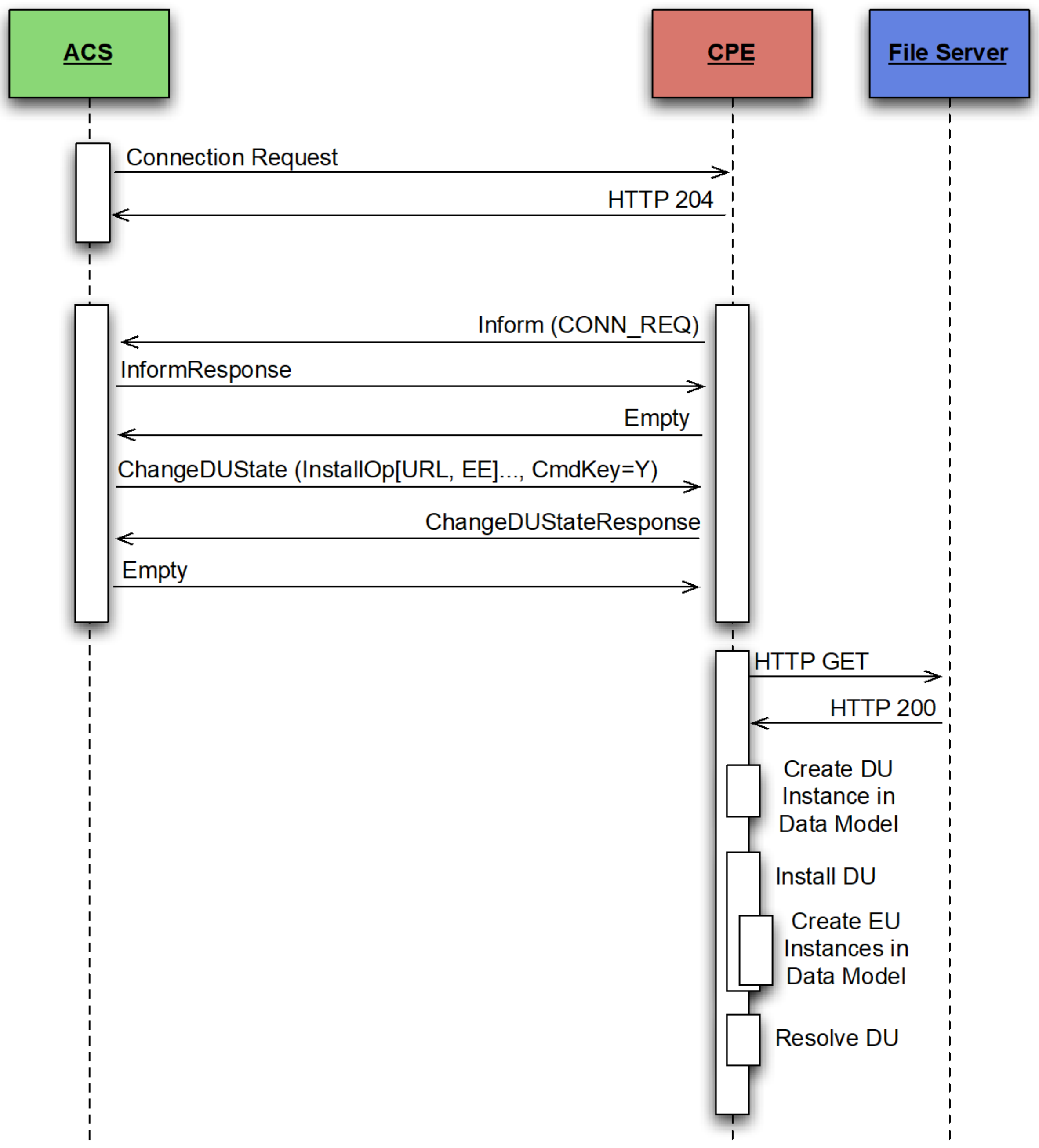


Figure 31 – Installation of a Deployment Unit - CWMP Session #1

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.

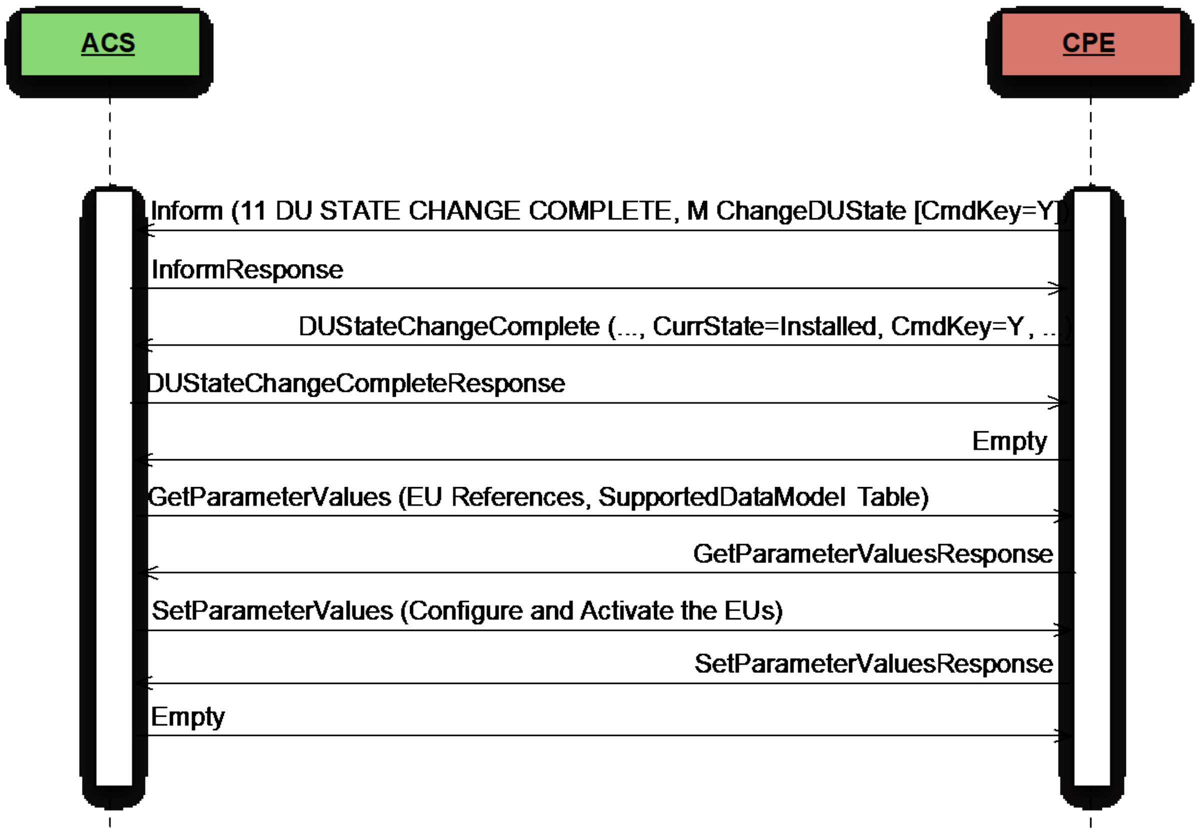

ACS
CPE
Inform (11 DU STATE CHANGE COMPLETE, M ChangeDUState [CmdKey=Y])
InformResponse
DUStateChangeComplete (..., CurrState=Installed, CmdKey=Y, ...)
DUStateChangeCompleteResponse
Empty
GetParameterValues (EU References, SupportedDataModel Table)
GetParameterValuesResponse
SetParameterValues (Configure and Activate the EUs)
SetParameterValuesResponse
Empty

Figure 32 – 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.

VI.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.

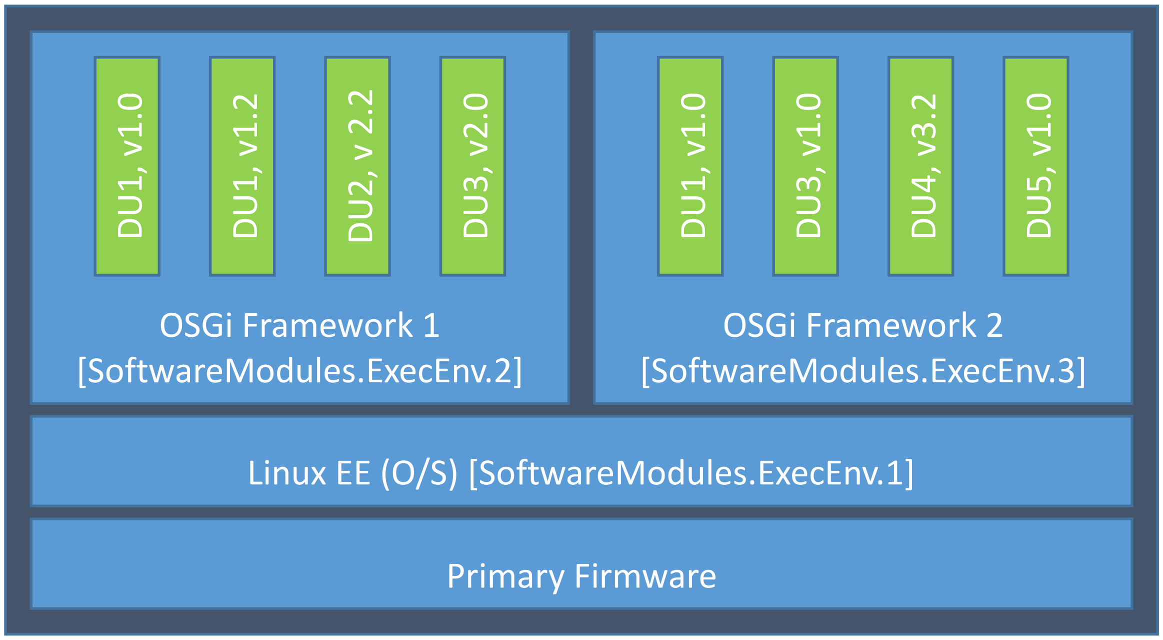


Figure 33 – Possible Multi-Execution Environment Implementation

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.

VI.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.

VI.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.

VI.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 [37] 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).

VI.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 [37] 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).

VI.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 [37] Version 3 (Name- Based).
- The DU caused an EE to come into existence on which at least 1 DU is Installed.

VI.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.

End of Broadband Forum Technical Report TR-069