TECHNICAL REPORT

TR-069

CPE WAN Management Protocol v1.1

Version: Issue 1 Amendment 1
Version Date: November 2006

Version History

Version Number	Version Date	Version 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

Technical comments or questions about this Broadband Forum Technical Report should be directed to:

Editors:	William Lupton	2Wire	wlupton@2wire.com
	John Blackford	2Wire	jblackford@2wire.com
	Mike Digdon	SupportSoft	mike.digdon@supportsoft.com
	Tim Spets	Westell	tspets@westell.com
BroadbandHome™ Technical Working Group Chair:	Greg Bathrick	PMC-Sierra	Greg_Bathrick@pmc-sierra.com
	Heather Kirksey	Motive	hkirksey@motive.com

Contents

Abstract:
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.

This document 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
- 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 Options using digitally signed Vouchers is defined in Annex C.

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

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.

The CPE WAN Management Protocol also defines a digitally signed file format that may optionally be used to download either individual files or a package of files along with explicit installation instructions for the CPE to perform. This signed package format ensures the integrity of downloaded files and the associated installation instructions, allowing authentication of a file source that may be a party other than the ACS operator.

1.1.3 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.

1.1.4 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.

1.1.5 Identity Management for Web Applications

To support web-based applications for access from a browser within the CPE's local network, the CPE WAN Management Protocol defines an optional mechanism that allows such web sites to customize their content with explicit knowledge of the associated CPE. This mechanism is described in Annex D.

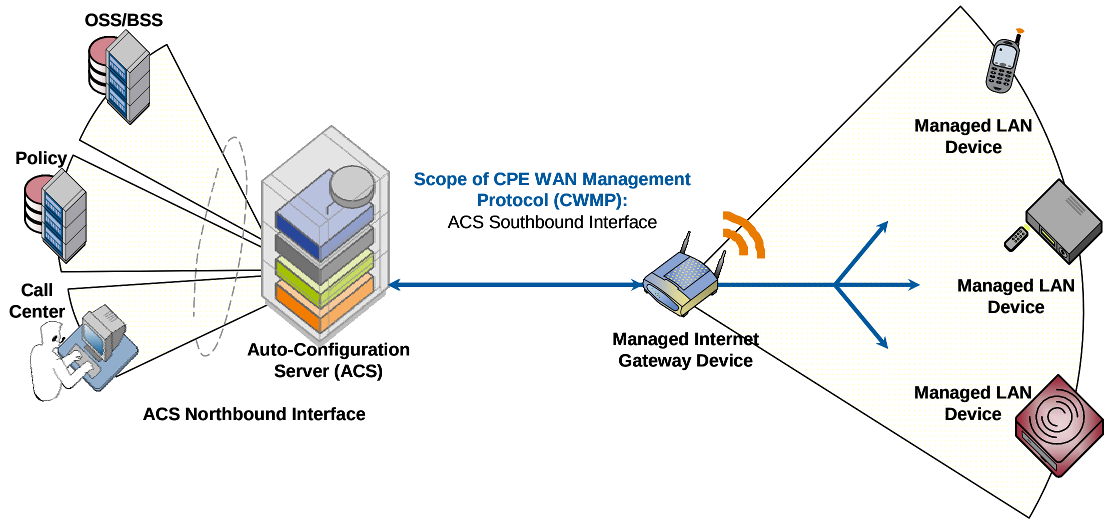
1.2 Positioning in the End-to-End Architecture

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 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.

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 CPE can interact with a single ACS at a time. At any time, a CPE 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.)

1.6 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.
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.
CPE	Customer Premises Equipment; refers to any TR-069-compliant device and therefore covers both Internet Gateway Devices and LAN-side end devices.
CWMP	CPE WAN Management Protocol (the subject of this standard).
Data Model	A hierarchical set of Parameters that define the managed objects accessible via TR-069 for a particular device or service.
Device	Used interchangeably with CPE.
Event	An indication that something of interest has happened that requires the CPE to notify the ACS.
Internet Gateway Device	A CPE device, typically a broadband router, that acts as a gateway between the WAN and the LAN.
Option	An optional CPE capability that may only be enabled or disabled using a digitally signed Voucher.
Parameter	A name-value pair representing a manageable CPE parameter made accessible to an ACS for reading and/or writing.
RPC	Remote Procedure Call.

[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.

Session	A contiguous sequence of CWMP transactions between a CPE and an ACS. Note that a Session may span multiple TCP connections.
STB	Set Top Box. This device contains Audio and Video decoders and is intended to be connected to Analog TV and / or Home Theaters.
Transaction	A message exchange between a CPE and ACS consisting of a single request followed by a single response, initiated either by the CPE or ACS.
Transaction Session	The same as a Session. The "Transaction" qualifier is sometimes used for emphasis.
VoIP Endpoint	A Voice over IP device that acts as the initiation/termination point for VoIP calls. Examples of Endpoints include VoIP phones and analog terminal adapters (ATAs).
Voucher	A digitally signed data structure that instructs a particular CPE to enable or disable Options, and characteristics that determine under what conditions the Options persist.

1.7 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].

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

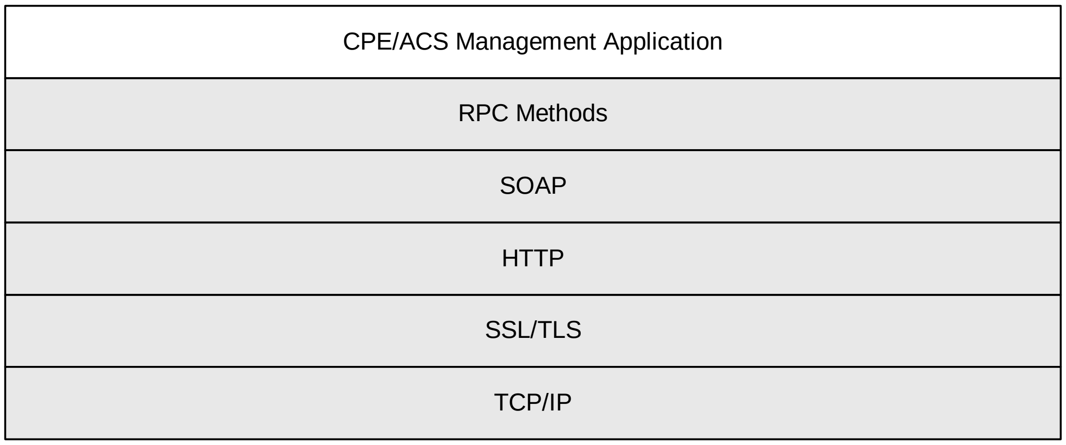


Table 1 – Protocol layer summary

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 [8].
HTTP	HTTP 1.1, as specified in [5].
SSL/TLS	The standard Internet transport layer security protocols. Specifically, either SSL 3.0 (Secure Socket Layer), as defined in [10], or TLS 1.0 (Transport Layer Security) as defined in [11].
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 SSL/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.

The protocol includes additional security mechanisms associated with the optional signed Voucher mechanism and the Signed Package Format, described in Annex C and Annex E, respectively.

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-106: Data Model Template for TR-069-Enabled Devices, [13]
- TR-098: Internet Gateway Device Data Model for TR-069, [24]
- TR-104: Provisioning Parameters for VoIP CPE, [25]

Each Parameter consists of a name-value pair. The name identifies the particular Parameter, and has a hierarchical structure similar to files in a directory, with each level separated by a "." (dot). The value of a Parameter may be one of several defined data types (see [13]).

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). 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 a mechanism to facilitate file downloads or (optionally) uploads for a variety of purposes, such as firmware upgrades or vendor-specific configuration files.

When initiated by the ACS, the CPE is provided with the location of the file to be transferred, using HTTP/HTTPS or, optionally, FTP, SFTP, or TFTP as the transport protocol. The CPE then performs the transfer, and notifies the ACS of the 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.

The CPE WAN Management Protocol also defines a digitally signed file format that may optionally be used for downloads. This Signed Package Format is defined in Annex E.

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 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 G, which accommodates CPE operating behind a NAT gateway 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 example, this MAY be done via a LAN-side CPE auto-configuration protocol. 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 [14]. The CPE would use DNS to resolve the IP address of the ACS from the host name component of the URL. In this case a second DHCP option MAY be used to set the ProvisioningCode, which MAY be used to indicate the primary service provider and other provisioning information to the ACS.

   A CPE identifies itself to the DHCP server as supporting this method by including the string “broadband-forum.org” (all lower case) anywhere in the Vendor Class Identifier (DHCP option 60).

   The CPE MAY use the values received from the DHCP server in the Vendor Specific Information (DHCP option 43) to set the corresponding parameters as listed in Table 2. This DHCP option is encoded as a list of one or more Encapsulated Vendor-Specific Options in the format defined in [14]. This list MAY include other vendor-specific options in addition to those listed here.

   If the CPE obtained an ACS URL through DHCP and it cannot reach the ACS, the CPE MUST perform a DHCP Inform 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 reply for the DHCP Inform, it MUST attempt to retry according to RFC 2131.

   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 default) 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 the first time it successfully contacted it. 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]
URL of the ACS	1	...ManagementServer.URL
Provisioning code	2	...DeviceInfo.ProvisioningCode

The specified URL MUST be an absolute URL. Both the URL and ProvisioningCode MUST NOT be null terminated. If the CPE receives a URL or ProvisioningCode value that is null terminated, the CPE MUST accept the value provided, and MUST NOT interpret the null character as part of the URL or ProvisioningCode 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 [5]. 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, the ACS MAY at any time modify the ACS address Parameter stored within the CPE (...ManagementServer.URL, as defined in [13]). Once modified, the CPE MUST use the modified address 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 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 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 along 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

[2] As defined in [13].

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 [17]), 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 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) 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 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 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.

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.

Beginning with the tenth post-reboot session retry attempt, the CPE MUST choose from a range between 2560 and 5120 seconds. The CPE MUST continue to retry a failed session until it is successfully terminated. 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	**Wait interval range (min-max seconds)**
#1	5-10
#2	10-20
#3	20-40
#4	40-80
#5	80-160
#6	160-320
#7	320-640
#8	640-1280
#9	1280-2560
#10 and subsequent	2560-5120

3.2.2 ACS Connection Initiation

The ACS MAY at any time request that the CPE 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 having an IP address that is routable from the ACS. If the CPE is behind a firewall or NAT device lying between the ACS and CPE, the ACS might not be able to access the CPE at all. Annex G defines a mechanism that allows an ACS to contact a CPE connected via a NAT device.

The Connection Request mechanism is defined as follows:

- 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 digest-authentication to authenticate the ACS before proceeding—the CPE MUST NOT initiate a connection to the ACS due to an unsuccessfully authenticated request.
- The CPE MUST accept Connection Requests from any source that has the correct authentication parameters for the target CPE.
- 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 in the HTTP response MUST be zero.

- The CPE SHOULD restrict the number of Connection Requests 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). In this case, the CPE SHOULD NOT include the HTTP Retry-After header in the response.
- If the CPE successfully authenticates and responds to a Connection Request as described above, and if it is not already in a session, 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, once the CPE successfully authenticates and responds to a Connection Request, but before it establishes a session to the ACS, it receives one or more successfully authenticated Connection Requests, the CPE MUST return a successful response for each of those Connection Requests, but MUST NOT initiate any additional sessions 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 when it receives one or more Connection Requests, it MUST NOT terminate that session prematurely as a result. The CPE MUST instead take one of the following 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 session, initiate exactly one new session (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. In this case, the CPE MUST initiate the session immediately after the existing session is complete and all changes from that session have been applied.

  This requirement holds for Connection Requests received any time during the interval that the CPE considers itself in a session, 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.

This mechanism relies on the ACS having had at least one prior communication with the CPE via 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 [13]). If the URL used for management access changes, the CPE MUST notify the ACS by issuing an Inform message indicating the new management IP address (see [13]), thus keeping the ACS up-to-date.

Port 7547 has been assigned by IANA for the CPE WAN Management Protocol (see [17]), and the CPE MAY use this port in the Connection Request URL.

3.3 Use of SSL/TLS and TCP

The use of SSL/TLS to transport the CPE WAN Management Protocol is RECOMMENDED, although the protocol MAY be used directly over a TCP connection instead. If SSL/TLS is not used, some aspects of security are sacrificed. Specifically, SSL/TLS provides confidentiality and data integrity, and allows certificate-based authentication in lieu of shared secret-based authentication.

Certain restrictions on the use of SSL/TLS and TCP are defined as follows:

- The CPE MUST support SSL 3.0 [10], TLS 1.0 [11] or both.
- If CPE supports both, it SHOULD communicate both capabilities to the ACS as specified in Appendix E of RFC 2246, allowing the ACS to choose the protocol.
- If the ACS URL has been specified as an HTTPS URL, the CPE MUST establish connections to the ACS using SSL / TLS.
- The CPE MUST support the following SSL / TLS cipher suites:
    - RSA_WITH_3DES_EDE_CBC_SHA
    - RSA_WITH_RC4_128_SHA
- A CPE MUST be able to initiate outgoing connections to the ACS.
- An ACS MUST be able to accept CPE-initiated connections.
- If SSL/TLS 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, and that the CPE MUST ensure that the value of the CN (Common Name) component of the Subject field in the certificate exactly matches the host portion of the ACS URL known to the CPE (even if the host portion of the ACS URL is an IP address). This MUST be a direct string comparison between the CN and the host portion of the ACS URL. If either of these is in the form of a hostname (rather than an IP address), this comparison MUST NOT involve the IP address that the hostname resolves to.

  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 CN component of the ACS certificate against the host portion of the redirected ACS URL rather than the pre-configured ACS URL.

  A CPE SHOULD wait until it has accurate absolute time before contacting the ACS. 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 [5], 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 [8], 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).

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:broadband-forum-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.

3.4.2 Transaction 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. 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.

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.[3] 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 [7]. The ACS MAY make use of old-style "Netscape" cookies as well as, or instead of, the new-style cookies of [7]. 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 [7] 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. In particular, because the ACS might send old-style, new-style, or a mixture of old-style and new-style cookies, the CPE MUST support the compatibility requirements of section 9.1 of [7]. The CPE MUST support the use of multiple cookies by the ACS, and MUST make available at least 512 bytes for storage of cookies.

When a transaction session is completed successfully or terminated unsuccessfully, a CPE MUST close the associated TCP connection to the ACS and discard all cookies marked for *Discard*.

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.
- 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, 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 ...ManagementServer.URL, as defined in [13]) 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 SSL/TLS) MUST be used with the new target regardless of the transport used before redirection.
- If SSL/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).

[3] 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.

- In an HTTP response sent by the ACS containing a redirect status code, the length of the HTTP message-body MUST be zero. If the CPE receives an HTTP re-direct response with a non-empty message-body, it MUST ignore the content of the message-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 [7], 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 the Download or Upload 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.
- 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 SSL / TLS as specified in section 3.3 for establishment of a separate TCP connection to transfer a file using HTTP. The CPE MUST use SSL / TLS when the file location is specified as an HTTPS URL.

The CPE MUST support both HTTP basic and digest authentication 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, Upload).

3.4.4 Authentication

If the CPE is not authenticated using SSL/TLS, the ACS MUST authenticate the CPE using HTTP. If SSL/TLS is being used for encryption, the ACS MAY use either basic or digest authentication [6]. If SSL/TLS is not being used, then the ACS MUST use digest authentication.

The CPE MUST support both HTTP basic and digest authentication. The ACS chooses the authentication scheme by virtue of providing basic or digest authentication challenge.

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 i,n the <ProductClass> and <SerialNumber> that is not either alphanumeric or an underscore (“_”) MUST be escaped using URI percent encoding, as defined in RFC 3986 [12].

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 SSL/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 SSL/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 addional HTTP-related requirements are specified:

- Whenever the ACS sends an empty HTTP response, it MUST use the “204 (No Content)” HTTP status code.
- Whenever the CPE sends an empty HTTP request, the length of the HTTP message-body MUST be zero.
- The CPE MUST NOT make use of pipelining as defined in HTTP 1.1 [5].

3.5 Use of SOAP

The CPE WAN Management Protocol defines SOAP 1.1 [8] 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/"
- All elements and attributes defined as part of this version of the CPE WAN Management Protocol are associated with the following namespace identifier:
    - "urn:broadband-forum-org:cwmp-1-0"
- 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 [8], 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 [8]), 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 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 defined in the “urn:broadband-forum-org:cwmp-1-0” namespace. 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:broadband-forum-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:broadband-forum-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>
                            InternetGatewayDevice.Time.LocalTimeZone
                        </ParameterName>
                        <FaultCode>9012</FaultCode>
                        <FaultString>Not a valid time zone value</FaultString>
                    </SetParameterValuesFault>
                    <SetParameterValuesFault>
                        <ParameterName>
                            InternetGatewayDevice.Time.LocalTimeZoneName
                        </ParameterName>
                        <FaultCode>9012</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.

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[4] 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/XMLSchema. The recipient MAY reject an RPC method that references a different version of either of these namespaces.

[4] With the exception that reception of an unknown SOAP action MUST result in a fault response indicating Method Not Supported (see Annex A).

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

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 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.

Below is an example of a message showing the use of all of the defined headers:

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:cwmp="urn:broadband-forum-org:cwmp-1-0">
    <soap:Header>
        <cwmp:ID soap:mustUnderstand="1">1234</cwmp:ID>
        <cwmp:HoldRequests soap:mustUnderstand="1">0</cwmp:HoldRequests>
    </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.

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[5]	REQUIRED[5]
`Upload`	OPTIONAL	OPTIONAL
`FactoryReset`	OPTIONAL	OPTIONAL
`GetQueuedTransfers`	OPTIONAL	OPTIONAL
`ScheduleInform`	OPTIONAL	OPTIONAL
`SetVouchers`	OPTIONAL[6]	OPTIONAL[6]
`GetOptions`	OPTIONAL[6]	OPTIONAL[6]
ACS methods	Calling	Responding
`GetRPCMethods`	OPTIONAL	REQUIRED
`Inform`	REQUIRED	REQUIRED
`TransferComplete`	REQUIRED[7]	REQUIRED[8]
`RequestDownload`	OPTIONAL	OPTIONAL
`Kicked`	OPTIONAL	OPTIONAL

3.7 Transaction Session Procedures

All transaction 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

[5] Required only if file downloads of any type are supported.
[6] If the voucher mechanism is supported, both the SetVouchers and GetOptions methods are required.
[7] Required only if file downloads or uploads of any type are supported.
[8] Required only if the ACS supports initiation of file downloads or uploads.

"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 transaction session between a CPE and its associated ACS can exist at a time.

> *Note – a transaction 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 transaction 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.

The CPE MUST consider the session to have been successfully initiated if and only if it receives a successful Inform response.

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:

- The most recently received HTTP response from the ACS did not contain a SOAP request.
- The ACS has indicated that HoldRequests is false (see section 3.5). 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).
- 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 3.5), 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 6 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 6 – CPE Message Transmission Constraints

	HoldRequests	**ACS request outstanding**	**No ACS request outstanding**
CPE requests pending[9]	**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 transaction 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 6, 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.
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.

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

[9] 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.

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. 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.

For most events, delivery is confirmed when the CPE receives a successful InformResponse. Three standard event types (KICKED, TRANSFER COMPLETE, REQUEST DOWNLOAD) indicate that one or more methods (Kicked [section A.4.2.1], TransferComplete [section A.3.3.2], RequestDownload [section A.4.2.2] 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 7), 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 7 – 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 CPE to the ACS from the factory. • First time connection of the CPE to the ACS after a factory reset. • First time connection of the CPE 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.
"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.	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.
"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.
"4 VALUE CHANGE"	Single	Indicates that since the last successful 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 or the ACS URL is modified before discarding it.

Event Code	Cumulative Behavior	Explanation	ACS Response for Successful Delivery	Retry/Discard Policy
"5 KICKED"	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.
"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" and/or "M Upload" event codes (see "M Download" and "M Upload", below).	TransferCompleteResponse	The CPE MUST NOT ever discard an undelivered TRANSFER COMPLETE event.
"8 DIAGNOSTICS COMPLETE"	Single	Used when reestablishing a connection to the ACS after completing one or more diagnostic test initiated by the ACS.	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.
"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.
"M ScheduleInform"	Multiple	The ACS requested a scheduled Inform.	InformResponse	The CPE MUST NOT ever discard an undelivered "M ScheduleInform" event.
"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.
"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.
"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

Event Code	Cumulative Behavior	Explanation	ACS Response for Successful Delivery	Retry/Discard Policy
"X "<OUI> " " <event>	Not specified	Vendor-specific event. The OUI after the "X" and space is an organizationally unique identifier 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 [9], and MUST be one that is assigned to the organization that defined this vendor-specific event. The value and interpretation of <event> is vendor-specific. For example: "X 012345 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. 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 Inform response.

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.

If the ACS wishes to prevent the CPE sending requests during some portion of the session, it MAY do so by setting the HoldRequests header to true in each envelope transmitted to the CPE until the ACS again wishes to allow requests from the CPE. The ACS MUST allow CPE requests before completion of a session (this MAY be done either explicitly via the HoldRequests header or implicitly by sending an empty HTTP response).

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 8 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 8 – 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 all 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.

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, 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.

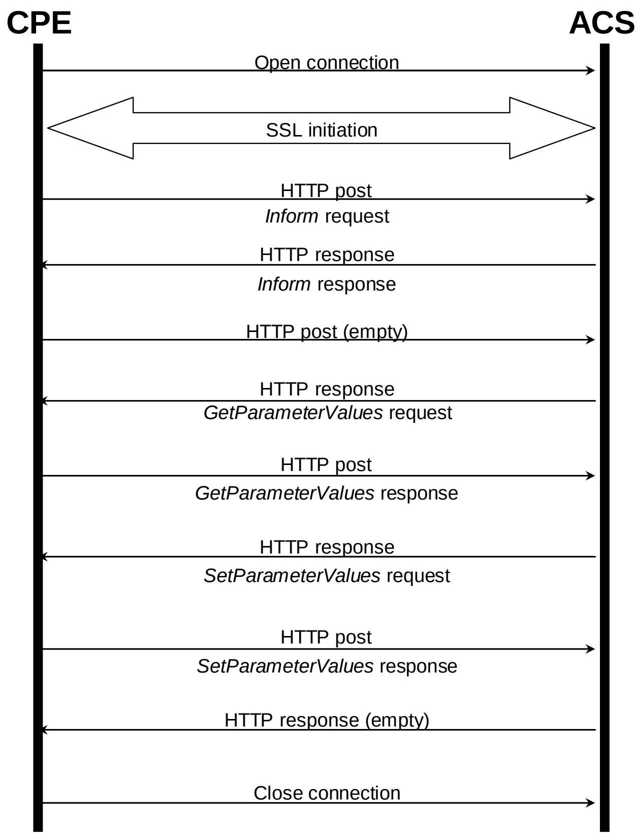
If the ACS receives a SOAP fault response from the CPE, as defined in section 3.5, the ACS MAY choose among the following actions:

- 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.

3.7.3 Transaction Examples

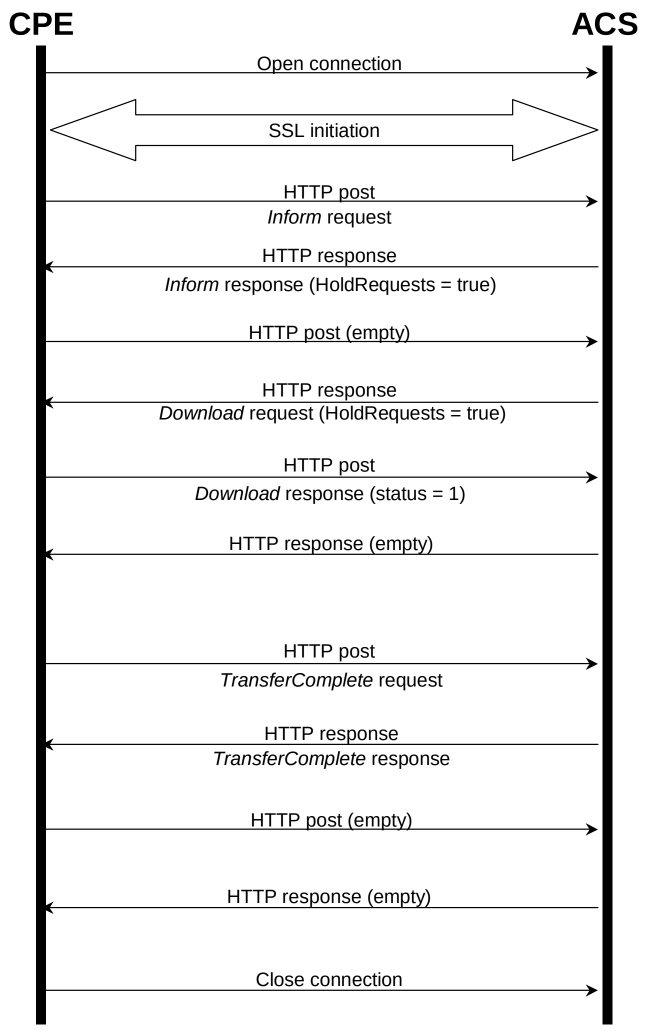
In the example shown in Figure 3, the ACS first reads a set of parameter values, and based on the result, sets some parameter values.

Figure 3 – Transaction Session Example



In the example shown in Figure 4, the ACS first initiates a file download, and the CPE sends a TransferComplete later in the same session. Note that this scenario could occur only if the file download is very short and the CPE is capable of performing it in parallel with the ongoing CPE WAN Management Protocol session (which a CPE is *not* required to do). To allow this possibility, the ACS sets HoldRequests equal to true until it has completed sending requests to the CPE.

Figure 4 – Example with the ACS using HoldRequests equal true



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*, The 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*, The Broadband Forum Technical Report

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

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

[5] RFC 2616, *Hypertext Transfer Protocol -- HTTP/1.1*, http://www.ietf.org/rfc/rfc2616.txt

[6] RFC 2617, *HTTP Authentication: Basic and Digest Access Authentication*, http://www.ietf.org/rfc/rfc2617.txt

[7] RFC 2965, *HTTP State Management Mechanism*, http://www.ietf.org/rfc/rfc2965.txt

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

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

[10] *The SSL Protocol, Version 3.0*, http://wp.netscape.com/eng/ssl3

[11] RFC *2246, The TLS Protocol, Version 1.0*, http://www.ietf.org/rfc/rfc2246.txt

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

[13] TR-106 Amendment 1, *Data Model Template for TR-069-Enabled Devices*, The Broadband Forum Technical Report

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

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

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

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

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

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

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

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

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

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

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

[24] TR-098 Amendment 1, *Internet Gateway Device Data Model for TR-069*, The Broadband Forum Technical Report

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

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 *version 1* of the specific procedure calls (methods) that are defined. 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 3.5.

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 [8]. The complete set of types utilized in this specification along with the notation used to represent these types is listed in Table 9.

Table 9 – 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 timezoned representation (that is, it MUST include a timezone suffix).
base64	Base64 encoded binary. A maximum allowed length can be listed using the form base64(N), where N is the maximum length in characters after 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 [8], elements specified as being of type "anySimpleType" MUST include a type attribute to indicate the actual type of the element. For example:
	`<ParameterValueStruct>`
	`<Name>InternetGatewayDevice.ProvisioningCode</Name>`
	`<Value xsi:type="xsd:string">code12345</Value>`
	`</ParameterValueStruct>`
	The namespaces xsi and xsd used above are as defined in [8].

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 would be 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 Other Requirements

All methods MUST be called using the exact number of arguments specified in this document. Methods called with either missing arguments or extra arguments MUST generate an error response. Argument order MUST be as specified in this document.

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

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 [9], 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.

The calling arguments for this method are defined in Table 10. The arguments in the response are defined in Table 11.

Table 10 – GetRPCMethods arguments

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

Table 11 – 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 12. The arguments in the response are defined in Table 13.

Table 12 – SetParameterValues arguments

Argument	Type	Description
`ParameterList`	ParameterValueStruct[]	Array of name-value pairs as specified in Table 14. 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 and no fault response is generated. 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 13 – 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).

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—the application of value changes to the CPE MUST be independent from the order in which they are listed.

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. 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 14.

Table 14 – 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 following fault codes are defined for this method: 9001, 9002, 9003, 9004, 9005, 9006, 9007, 9008.

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

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 and are not required by the data model(s) or profiles (as defined in [13]) 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 15. The arguments in the response are defined in Table 16.

Table 15 – 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. Below is an example of a full Parameter name: InternetGatewayDevice.DeviceInfo.SerialNumber Below is an example of a partial path name: InternetGatewayDevice.DeviceInfo.

Table 16 – GetParameterValuesResponse arguments

Argument	Type	Description
`ParameterList`	ParameterValueStruct[]	Array of name-value pairs, as specified in Table 14, 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.

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).

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 17. The arguments in the response are defined in Table 18.

Table 17 – 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. Below is an example of a full Parameter name: InternetGatewayDevice.DeviceInfo.SerialNumber Below is an example of a partial path name: InternetGatewayDevice.DeviceInfo.

Argument	Type	Description
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 "InternetGatewayDevice.LANDevice.1.Hosts.", the response would include the following (if there were a single instance of Host with instance number "1"): InternetGatewayDevice.LANDevice.1.Hosts. InternetGatewayDevice.LANDevice.1.Hosts.HostNumberOfEntries InternetGatewayDevice.LANDevice.1.Hosts.Host. InternetGatewayDevice.LANDevice.1.Hosts.Host.1. InternetGatewayDevice.LANDevice.1.Hosts.Host.1.IPAddress InternetGatewayDevice.LANDevice.1.Hosts.Host.1.AddressSource InternetGatewayDevice.LANDevice.1.Hosts.Host.1.LeaseTimeRemaining InternetGatewayDevice.LANDevice.1.Hosts.Host.1.MACAddress InternetGatewayDevice.LANDevice.1.Hosts.Host.1.HostName InternetGatewayDevice.LANDevice.1.Hosts.Host.1.InterfaceType InternetGatewayDevice.LANDevice.1.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 "InternetGatewayDevice.LANDevice.1.Hosts.", the response would include the following: InternetGatewayDevice.LANDevice.1.Hosts.HostNumberOfEntries InternetGatewayDevice.LANDevice.1.Hosts.Host. Or, if ParameterPath were empty, with NextLevel equal true, the response would list only "InternetGatewayDevice." (if the CPE is an Internet Gateway Device).

Table 18 – 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 19. 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 `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. 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, 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 19 – ParameterInfoStruct definition

Name	Type	Description
Name	string(256)	This is the full path name of a Parameter or a partial path.

Name	Type	Description
`Writable`	boolean	Whether or not the Parameter value can be overwritten using the SetParameterValues method. If Name is a partial path 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 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, 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, the CPE MUST return a fault response with the Invalid Arguments fault code (9003).

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 20. The arguments in the response are defined in Table 21.

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.

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

Table 20 – 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 22. As described above, the order of entries in this array is significant.

Table 21 – SetParameterAttributesResponse arguments

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

Table 22 – 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 number 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: InternetGatewayDevice.DeviceInfo.SerialNumber Below is an example of a partial path name: InternetGatewayDevice.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[0:2]	Indicates whether the CPE will include changed values of the specified parameter(s) in the Inform message, and whether the CPE will initiate a session to 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 Inform message that is sent the next time a session is established to the ACS. If the CPE has rebooted, or the URL of the ACS has changed since the last session, the CPE MAY choose not to include the list of changed parameters in the first session established with the new 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. For parameters defined in the corresponding data

Name	Type	Description
		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. 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 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. 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 [13]. 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 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 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).

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 23. The arguments in the response are defined in Table 24.

Table 23 – 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. Below is an example of a full Parameter name: InternetGatewayDevice.DeviceInfo.SerialNumber Below is an example of a partial path name: InternetGatewayDevice.DeviceInfo.

Table 24 – 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 the ParameterNames argument in the request was a partial path, and if there are no Parameters within the object represented by that partial path (at any level below), the ParameterList MUST be empty, and this MUST NOT cause an error response.

Table 25 – 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.
`Notification`	int[0:2]	Indicates whether the CPE will include changed values of the specified parameter(s) in the Inform message, and whether the CPE will initiate a session to 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 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.

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. 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 [13]. 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).

A.3.2.6 AddObject

This method MAY be used by the ACS to create a new instance of a multi-instance object—a collection of Parameters and/or other objects for which multiple instances are defined. 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 number. 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
```

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 26. The arguments in the response are defined in Table 27.

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 numbers.

Table 26 – 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.
`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 and no fault response is generated. 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 27 – 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 in the path name. The instance number assigned by the CPE is arbitrary and instance numbers assigned by sequential calls to AddObject need not be consecutive. 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 number 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.
`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, 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.

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 number. 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 28. The arguments in the response are defined in Table 29.

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 28 – 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 number 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 and no fault response is generated. 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 29 – 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

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 30. The arguments in the response are defined in Table 31.

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 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 that the downloaded file is a software image, the CPE MUST consider the downloaded file 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.

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 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 30 – 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 and GetQueuedTransfers. 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” The following format is defined to allow the unique definition of vendor-specific file types: "X <OUI> <Vendor-specific identifier>" <OUI> is replaced by a 6 hexadecimal-digit OUI (organizationally unique identifier) as defined in [9], with all upper-case letters and any leading zeros included. The OUI used for a given vendor-specific file type MUST be one that is assigned to the organization that defined this file type (which is not necessarily the same as the vendor of the CPE or ACS). 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. 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".
`URL`	string(256)	URL, as defined in [12], 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 [12].
`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.
`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.

Argument	Type	Description
`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.
`DelaySeconds`	unsignedInt	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 transaction 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 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 [12], 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 [12], 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 31 – 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.

Argument	Type	Description
`CompleteTime`	dateTime	The date and time 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 32. The arguments in the response are defined in Table 33.

> *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. 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.

Table 32 – 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 33 – 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 34. The arguments in the response are defined in Table 35.

Table 34 – Inform arguments

Argument	Type	Value
`DeviceId`	DeviceIdStruct	A structure that uniquely identifies the CPE, defined in Table 36.
`Event`	EventStruct[64]	An array of structures, as defined in Table 7 in section 3.7.1.5, indicating the events that caused the transaction 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.
`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.

Argument	Type	Value
`ParameterList`	ParameterValueStruct[]	Array of name-value pairs as specified in Table 14. 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 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 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 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 35 – 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 36 – DeviceIdStruct definition

Name	Type	Description
`Manufacturer`	string(64)	Manufacturer of the device (for display only).
`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 [9].

Name	Type	Description
`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.

Table 37 – 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 7 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 7 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: • ScheduledInform method (EventCode = “M ScheduleInform”) • Reboot method (EventCode = “M Reboot”) • Download method (EventCode = “M Download”) • 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.

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

This method informs the ACS of the completion (either successful or unsuccessful) of a file transfer initiated by an earlier Download or Upload method call. This 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 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 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 38. The arguments in the response are defined in Table 39.

Table 38 – TransferComplete arguments

Argument	Type	Value
`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.
`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 transfer completed 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 39 – TransferCompleteResponse arguments

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

Table 40 – 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. 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.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

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 41. The arguments in the response are defined in Table 42.

Table 41 – GetQueuedTransfers arguments

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

Table 42 – GetQueuedTransfersResponse arguments

Argument	Type	Description
`TransferList`	QueuedTransferStruct[16]	Array of structures as defined in Table 43, each describing the state of one transfer that the CPE has been instructed to perform, but has not yet been fully completed.

Table 43 – 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 44. The arguments in the response are defined in Table 45.

Table 44 – 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 intiate 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 45 – 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

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 46. The arguments in the response are defined in Table 47.

Table 46 – 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 47 – 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

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 48. The arguments in the response are defined in Table 49.

Table 48 – 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 49 – GetOptionsResponse arguments

Argument	Type	Description
`OptionList`	OptionStruct[]	Array of OptionStructs as defined in Table 50, 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 50 – 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.

Name	Type	Description
`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 51. The arguments in the response are defined in Table 52.

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 51 – 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 and GetQueuedTransfers. 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 Vendor Configuration File” “2 Vendor Log File” The following format is defined to allow the unique definition of vendor-specific file types: "X <OUI> <Vendor-specific identifier>" <OUI> is replaced by a 6 hexadecimal-digit OUI (organizationally unique identifier) as defined in [9], with all upper-case letters and any leading zeros included. The OUI used for a given vendor-specific file type MUST be one that is assigned to the organization that defined this file type (which is not necessarily the same as the vendor of the CPE or ACS). 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.

Argument	Type	Description
`URL`	string(256)	URL, as defined in [12], 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 [12].
`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 transaction session in which the request was issues. 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 52 – 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 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 53. The arguments in the response are defined in Table 54.

Table 53 – FactoryReset arguments

Argument	Type	Description
-	void	This method has no arguments.

Table 54 – 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.4.2 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

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 55. The arguments in the response are defined in Table 56.

Table 55 – 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 56 – 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 57. The arguments in the response are defined in Table 58.

Table 57 – RequestDownload arguments

Argument	Type	Value
`FileType`	string(64)	This is the FileType being requested (see Table 30 for the list of allowed file types).
`FileTypeArg`	ArgStruct[16]	Array of zero or more additional arguments, where each argument is a structure of name-value pairs as defined in Table 59. The use of the additional arguments depend on the FileType specified. The following arguments are defined for each of the currently defined file types. 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.

FileType	FileTypeArg Names
1 Firmware Upgrade	(none)
2 Web Content	"Version"
3 Vendor Configuration File	(none)

Table 58 – RequestDownloadResponse arguments

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

Table 59 – 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.5 Fault Handling

A.5.1 CPE Fault Codes

Table 60 lists the fault codes that can be returned by a CPE. Note that the fault code values are shown in decimal representation.

Table 60 – Fault codes

Fault code	Description	Type[10]
`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
`9005`	Invalid parameter name (associated with Set/GetParameterValues, GetParameterNames, Set/GetParameterAttributes, AddObject, and DeleteObject)	Client

[10] The specified Type MUST be used to determine the value of the SOAP faultcode element as described in section 3.5.

Fault code	Description	Type[10]
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	Download failure (associated with Download or TransferComplete methods).	Server
9011	Upload failure (associated with Upload or TransferComplete methods).	Server
9012	File transfer server authentication failure (associated with Upload, Download, or TransferComplete methods).	Server
9013	Unsupported protocol for file transfer (associated with Upload and Download methods).	Server
9800 – 9899	Vendor defined fault codes	-

A.5.2 ACS Fault Codes

Table 61 lists the fault codes that can be returned by an ACS. Note that the fault code values are shown in decimal representation.

Table 61 – Fault codes

Fault code	Description	Type[10]
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
8800 – 8899	Vendor defined fault codes	-

A.6 RPC Method XML Schema

The XML schema for all RPC methods defined for the CPE WAN Management Protocol is specified below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  TR069 xml schema definition

  Define the xml schema for the messages passed in TR069
-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
           xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
           xmlns:tns="urn:broadband-forum-org:cwmp-1-0"
           targetNamespace="urn:broadband-forum-org:cwmp-1-0"
           elementFormDefault="unqualified"
           attributeFormDefault="unqualified">

  <xs:import namespace="http://schemas.xmlsoap.org/soap/envelope/"
             schemaLocation="http://schemas.xmlsoap.org/soap/envelope/"/>
  <xs:import namespace="http://schemas.xmlsoap.org/soap/encoding/"
             schemaLocation="http://schemas.xmlsoap.org/soap/encoding/"/>


  <!--
       SOAP Header Elements
  -->
  <xs:element name="ID">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute ref="soapenv:mustUnderstand" use="required" fixed="1"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

  <xs:element name="HoldRequests">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:boolean">
          <xs:attribute ref="soapenv:mustUnderstand" use="required" fixed="1"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>


  <!--
       SOAP Fault Extensions
  -->
  <xs:simpleType name="FaultCodeType">
    <xs:union>
      <xs:simpleType>
        <xs:restriction base="xs:unsignedInt">
          <xs:annotation>
            <xs:documentation>CPE fault codes</xs:documentation>
          </xs:annotation>
          <xs:enumeration value="9000">
            <xs:annotation>
              <xs:documentation>Method not supported</xs:documentation>
            </xs:annotation>
          </xs:enumeration>
          <xs:enumeration value="9001">
            <xs:annotation>
              <xs:documentation>Request denied (no reason specified)</xs:documentation>
            </xs:annotation>
          </xs:enumeration>
          <xs:enumeration value="9002">
            <xs:annotation>
              <xs:documentation>Internal error</xs:documentation>
            </xs:annotation>
          </xs:enumeration>
          <xs:enumeration value="9003">
            <xs:annotation>
```

```
              <xs:documentation>Invalid arguments</xs:documentation>
            </xs:annotation>
          </xs:enumeration>
          <xs:enumeration value="9004">
            <xs:annotation>
              <xs:documentation>Resources exceeded</xs:documentation>
            </xs:annotation>
          </xs:enumeration>
          <xs:enumeration value="9005">
            <xs:annotation>
              <xs:documentation>Invalid parameter name</xs:documentation>
            </xs:annotation>
          </xs:enumeration>
          <xs:enumeration value="9006">
            <xs:annotation>
              <xs:documentation>Invalid parameter type</xs:documentation>
            </xs:annotation>
          </xs:enumeration>
          <xs:enumeration value="9007">
            <xs:annotation>
              <xs:documentation>Invalid parameter value</xs:documentation>
            </xs:annotation>
          </xs:enumeration>
          <xs:enumeration value="9008">
            <xs:annotation>
              <xs:documentation>Attempt to set a non-writable parameter</xs:documentation>
            </xs:annotation>
          </xs:enumeration>
          <xs:enumeration value="9009">
            <xs:annotation>
              <xs:documentation>Notification request rejected</xs:documentation>
            </xs:annotation>
          </xs:enumeration>
          <xs:enumeration value="9010">
            <xs:annotation>
              <xs:documentation>Download failure</xs:documentation>
            </xs:annotation>
          </xs:enumeration>
          <xs:enumeration value="9011">
            <xs:annotation>
              <xs:documentation>Upload failure</xs:documentation>
            </xs:annotation>
          </xs:enumeration>
          <xs:enumeration value="9012">
            <xs:annotation>
              <xs:documentation>File transfer server authentication failure</xs:documentation>
            </xs:annotation>
          </xs:enumeration>
          <xs:enumeration value="9013">
            <xs:annotation>
              <xs:documentation>Unsupported protocol for file transfer</xs:documentation>
            </xs:annotation>
          </xs:enumeration>
        </xs:restriction>
      </xs:simpleType>
      <xs:simpleType>
        <xs:restriction base="xs:unsignedInt">
          <xs:annotation>
            <xs:documentation>CPE Vendor fault codes</xs:documentation>
          </xs:annotation>
          <xs:minInclusive value="9800"/>
          <xs:maxInclusive value="9899"/>
        </xs:restriction>
      </xs:simpleType>
      <xs:simpleType>
        <xs:restriction base="xs:unsignedInt">
          <xs:annotation>
            <xs:documentation>ACS fault codes</xs:documentation>
          </xs:annotation>
          <xs:enumeration value="8000">
            <xs:annotation>
              <xs:documentation>Method not supported</xs:documentation>
            </xs:annotation>
          </xs:enumeration>
          <xs:enumeration value="8001">
            <xs:annotation>
              <xs:documentation>Request denied (no reason specified)</xs:documentation>
            </xs:annotation>
          </xs:enumeration>
          <xs:enumeration value="8002">
            <xs:annotation>
              <xs:documentation>Internal error</xs:documentation>
            </xs:annotation>
          </xs:enumeration>
          <xs:enumeration value="8003">
            <xs:annotation>
              <xs:documentation>Invalid arguments</xs:documentation>
            </xs:annotation>
          </xs:enumeration>
          <xs:enumeration value="8004">
            <xs:annotation>
              <xs:documentation>Resources exceeded</xs:documentation>
            </xs:annotation>
          </xs:enumeration>
          <xs:enumeration value="8005">
            <xs:annotation>
              <xs:documentation>Retry request</xs:documentation>
            </xs:annotation>
          </xs:enumeration>
        </xs:restriction>
      </xs:simpleType>
      <xs:simpleType>
        <xs:restriction base="xs:unsignedInt">
          <xs:annotation>
            <xs:documentation>ACS Vendor fault codes</xs:documentation>
          </xs:annotation>
          <xs:minInclusive value="8800"/>
          <xs:maxInclusive value="8899"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:union>
  </xs:simpleType>
  <xs:element name="Fault">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="FaultCode" type="tns:FaultCodeType"/>
        <xs:element name="FaultString" type="xs:string" minOccurs="0"/>
        <xs:element name="SetParameterValuesFault" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ParameterName" type="xs:string"/>
              <xs:element name="FaultCode" type="tns:FaultCodeType"/>
              <xs:element name="FaultString" type="xs:string" minOccurs="0"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>


  <!--
       Type definitions used in messages
  -->
  <xs:complexType name="MethodList">
    <xs:complexContent>
      <xs:restriction base="soapenc:Array">
        <xs:sequence>
          <xs:element name="string" maxOccurs="unbounded">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:maxLength value="64"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
        </xs:sequence>
        <xs:attribute ref="soapenc:arrayType" use="required"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="FaultStruct">
    <xs:annotation>
      <xs:documentation>Fault information for TransferComplete</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="FaultCode">
        <xs:simpleType>
          <xs:restriction base="xs:unsignedInt">
            <xs:annotation>
              <xs:documentation>Transfer fault codes</xs:documentation>
            </xs:annotation>
            <xs:enumeration value="0">
              <xs:annotation>
                <xs:documentation>No fault</xs:documentation>
              </xs:annotation>
            </xs:enumeration>
            <xs:enumeration value="9001">
              <xs:annotation>
                <xs:documentation>Request denied (no reason specified)</xs:documentation>
              </xs:annotation>
            </xs:enumeration>
            <xs:enumeration value="9002">
              <xs:annotation>
                <xs:documentation>Internal error</xs:documentation>
              </xs:annotation>
            </xs:enumeration>
            <xs:enumeration value="9010">
              <xs:annotation>
                <xs:documentation>Download failure</xs:documentation>
              </xs:annotation>
            </xs:enumeration>
            <xs:enumeration value="9011">
              <xs:annotation>
                <xs:documentation>Upload failure</xs:documentation>
              </xs:annotation>
            </xs:enumeration>
            <xs:enumeration value="9012">
              <xs:annotation>
                <xs:documentation>File transfer server authentication failure</xs:documentation>
              </xs:annotation>
            </xs:enumeration>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="FaultString">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="256"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="DeviceIdStruct">
    <xs:sequence>
      <xs:element name="Manufacturer">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="64"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="OUI">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:length value="6"/>
            <xs:pattern value="[0-9A-F]{6}"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="ProductClass">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="64"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="SerialNumber">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="64"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="EventStruct">
    <xs:sequence>
      <xs:element name="EventCode">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="64"/>
            <xs:pattern value="0 BOOTSTRAP"/>
            <xs:pattern value="1 BOOT"/>
            <xs:pattern value="2 PERIODIC"/>
            <xs:pattern value="3 SCHEDULED"/>
            <xs:pattern value="4 VALUE CHANGE"/>
            <xs:pattern value="5 KICKED"/>
            <xs:pattern value="6 CONNECTION REQUEST"/>
            <xs:pattern value="7 TRANSFER COMPLETE"/>
            <xs:pattern value="8 DIAGNOSTICS COMPLETE"/>
            <xs:pattern value="9 REQUEST DOWNLOAD"/>
            <xs:pattern value="M Reboot"/>
            <xs:pattern value="M ScheduleInform"/>
            <xs:pattern value="M Download"/>
            <xs:pattern value="M Upload"/>
            <xs:pattern value="M X_\S+"/> <!-- no spaces in method names -->
            <xs:pattern value="X [0-9A-F]{6} .*"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="CommandKey" type="tns:CommandKeyType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="EventList">
    <xs:complexContent>
      <xs:restriction base="soapenc:Array">
        <xs:sequence>
          <xs:element name="EventStruct" type="tns:EventStruct" minOccurs="0" maxOccurs="64"/>
        </xs:sequence>
        <xs:attribute ref="soapenc:arrayType" use="required"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="ParameterValueStruct">
    <xs:sequence>
      <xs:element name="Name">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="256"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Value" type="xs:anySimpleType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ParameterValueList">
    <xs:complexContent>
      <xs:restriction base="soapenc:Array">
        <xs:sequence>
          <xs:element name="ParameterValueStruct" type="tns:ParameterValueStruct" minOccurs="0"
                      maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute ref="soapenc:arrayType" use="required"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="ParameterInfoStruct">
    <xs:sequence>
      <xs:element name="Name">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="256"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Writable" type="xs:boolean"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ParameterInfoList">
    <xs:complexContent>
      <xs:restriction base="soapenc:Array">
        <xs:sequence>
          <xs:element name="ParameterInfoStruct" type="tns:ParameterInfoStruct" minOccurs="0"
                      maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute ref="soapenc:arrayType" use="required"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="ParameterNames">
    <xs:complexContent>
      <xs:restriction base="soapenc:Array">
        <xs:sequence>
          <xs:element name="string" minOccurs="1" maxOccurs="unbounded">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:maxLength value="256"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
        </xs:sequence>
        <xs:attribute ref="soapenc:arrayType" use="required"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>

  <xs:simpleType name="ParameterKeyType">
    <xs:restriction base="xs:string">
      <xs:maxLength value="32"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="AccessList">
    <xs:complexContent>
      <xs:restriction base="soapenc:Array">
        <xs:sequence>
          <xs:element name="string" minOccurs="0" maxOccurs="unbounded">
            <xs:simpleType>
```

```
              <xs:restriction base="xs:string">
                <xs:maxLength value="64"/>
                <xs:enumeration value="Subscriber"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
        </xs:sequence>
        <xs:attribute ref="soapenc:arrayType" use="required"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="SetParameterAttributesStruct">
    <xs:sequence>
      <xs:element name="Name" nillable="true">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="256"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="NotificationChange" type="xs:boolean"/>
      <xs:element name="Notification">
        <xs:simpleType>
          <xs:restriction base="xs:int">
            <xs:enumeration value="0">
              <xs:annotation>
                <xs:documentation>Notification off. The CPE need not inform the ACS of a change
                     to the specified parameter(s)</xs:documentation>
              </xs:annotation>
            </xs:enumeration>
            <xs:enumeration value="1">
              <xs:annotation>
                <xs:documentation>Passive notification. Whenever the specified parameter value
                     changes, the CPE MUST include the new value in the ParameterList in the
                     Inform message that is sent the next time a session is established to the
                     ACS</xs:documentation>
              </xs:annotation>
            </xs:enumeration>
            <xs:enumeration value="2">
              <xs:annotation>
                <xs:documentation>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</xs:documentation>
              </xs:annotation>
            </xs:enumeration>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="AccessListChange" type="xs:boolean"/>
      <xs:element name="AccessList" type="tns:AccessList"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="SetParameterAttributesList">
    <xs:complexContent>
      <xs:restriction base="soapenc:Array">
        <xs:sequence>
          <xs:element name="SetParameterAttributesStruct" type="tns:SetParameterAttributesStruct"
                     minOccurs="1" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute ref="soapenc:arrayType" use="required"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="ParameterAttributeStruct">
    <xs:sequence>
      <xs:element name="Name">
        <xs:simpleType>
          <xs:restriction base="xs:string">
```

```
              <xs:maxLength value="256"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="Notification">
          <xs:simpleType>
            <xs:restriction base="xs:int">
              <xs:enumeration value="0"/>
              <xs:enumeration value="1"/>
              <xs:enumeration value="2"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="AccessList" type="tns:AccessList"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ParameterAttributeList">
      <xs:complexContent>
        <xs:restriction base="soapenc:Array">
          <xs:sequence>
            <xs:element name="ParameterAttributeStruct" type="tns:ParameterAttributeStruct"
                        minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute ref="soapenc:arrayType" use="required"/>
        </xs:restriction>
      </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="QueuedTransferStruct">
      <xs:sequence>
        <xs:element name="CommandKey" type="tns:CommandKeyType"/>
        <xs:element name="State">
          <xs:simpleType>
            <xs:restriction base="xs:int">
              <xs:enumeration value="1">
                <xs:annotation>
                  <xs:documentation>1 - Not yet started</xs:documentation>
                </xs:annotation>
              </xs:enumeration>
              <xs:enumeration value="2">
                <xs:annotation>
                  <xs:documentation>2 - In progress</xs:documentation>
                </xs:annotation>
              </xs:enumeration>
              <xs:enumeration value="3">
                <xs:annotation>
                  <xs:documentation>3 - Completed</xs:documentation>
                </xs:annotation>
              </xs:enumeration>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="TransferList">
      <xs:complexContent>
        <xs:restriction base="soapenc:Array">
          <xs:sequence>
            <xs:element name="QueuedTransferStruct" type="tns:QueuedTransferStruct" minOccurs="0"
                        maxOccurs="16"/>
          </xs:sequence>
          <xs:attribute ref="soapenc:arrayType" use="required"/>
        </xs:restriction>
      </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="VoucherList">
      <xs:complexContent>
        <xs:restriction base="soapenc:Array">
          <xs:sequence>
            <xs:element name="base64" type="soapenc:base64" minOccurs="1" maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute ref="soapenc:arrayType" use="required"/>
        </xs:restriction>
      </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="OptionStruct">
      <xs:sequence>
        <xs:element name="OptionName">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:maxLength value="64"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="VoucherSN" type="xs:unsignedInt"/>
        <xs:element name="State">
          <xs:simpleType>
            <xs:restriction base="xs:unsignedInt">
              <xs:enumeration value="0">
                <xs:annotation>
                  <xs:documentation>Option is disabled and not setup</xs:documentation>
                </xs:annotation>
              </xs:enumeration>
              <xs:enumeration value="1">
                <xs:annotation>
                  <xs:documentation>Option is enabled and not setup</xs:documentation>
                </xs:annotation>
              </xs:enumeration>
              <xs:enumeration value="2">
                <xs:annotation>
                  <xs:documentation>Option is disabled and setup</xs:documentation>
                </xs:annotation>
              </xs:enumeration>
              <xs:enumeration value="3">
                <xs:annotation>
                  <xs:documentation>Option is enabled and setup</xs:documentation>
                </xs:annotation>
              </xs:enumeration>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="Mode">
          <xs:simpleType>
            <xs:restriction base="xs:int">
              <xs:enumeration value="0">
                <xs:annotation>
                  <xs:documentation>0 - Disabled</xs:documentation>
                </xs:annotation>
              </xs:enumeration>
              <xs:enumeration value="1">
                <xs:annotation>
                  <xs:documentation>1 - Enabled with expiration</xs:documentation>
                </xs:annotation>
              </xs:enumeration>
              <xs:enumeration value="2">
                <xs:annotation>
                  <xs:documentation>2 - Enabled without expiration</xs:documentation>
                </xs:annotation>
              </xs:enumeration>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="StartDate" type="xs:dateTime"/>
        <xs:element name="ExpirationDate" type="xs:dateTime" minOccurs="0"/>
        <xs:element name="IsTransferable">
          <xs:simpleType>
            <xs:restriction base="xs:int">
              <xs:enumeration value="0">
                <xs:annotation>
                  <xs:documentation>Non-transferable</xs:documentation>
                </xs:annotation>
              </xs:enumeration>
              <xs:enumeration value="1">
                <xs:annotation>
                  <xs:documentation>Transferable</xs:documentation>
                </xs:annotation>
              </xs:enumeration>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="OptionList">
      <xs:complexContent>
        <xs:restriction base="soapenc:Array">
          <xs:sequence>
            <xs:element name="OptionStruct" type="tns:OptionStruct" minOccurs="0"
                        maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute ref="soapenc:arrayType" use="required"/>
        </xs:restriction>
      </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="ArgStruct">
      <xs:sequence>
        <xs:element name="Name">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:maxLength value="64"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="Value">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:maxLength value="256"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="FileTypeArg">
      <xs:complexContent>
        <xs:restriction base="soapenc:Array">
          <xs:sequence>
            <xs:element name="ArgStruct" type="tns:ArgStruct" minOccurs="0" maxOccurs="16"/>
          </xs:sequence>
          <xs:attribute ref="soapenc:arrayType" use="required"/>
        </xs:restriction>
      </xs:complexContent>
    </xs:complexType>

    <xs:simpleType name="CommandKeyType">
      <xs:restriction base="xs:string">
        <xs:maxLength value="32"/>
      </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="ObjectNameType">
      <xs:restriction base="xs:string">
        <xs:maxLength value="256"/>
        <xs:pattern value=".*\."/>
      </xs:restriction>
    </xs:simpleType>


    <!--
         Generic RPC Messages - Annex A.3.1
    -->
    <!-- GetRPCMethods -->
```

```
  <xs:element name="GetRPCMethods">
    <xs:annotation>
      <xs:documentation>GeRPCMethods message - Annex A.3.1.1</xs:documentation>
    </xs:annotation>
    <xs:complexType/>
  </xs:element>

  <!-- GetRPCMethodsResponse -->
  <xs:element name="GetRPCMethodsResponse">
    <xs:annotation>
      <xs:documentation>GeRPCMethodsResponse message - Annex A.3.1.1</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="MethodList" type="tns:MethodList"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>


  <!--
       CPE messages - Annex A.3.2
  -->
  <!-- SetParameterValues -->
  <xs:element name="SetParameterValues">
    <xs:annotation>
      <xs:documentation>SetParameterValues message - Annex A.3.2.1</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ParameterList" type="tns:ParameterValueList"/>
        <xs:element name="ParameterKey" type="tns:ParameterKeyType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- SetParameterValuesResponse -->
  <xs:element name="SetParameterValuesResponse">
    <xs:annotation>
      <xs:documentation>SetParameterValuesResponse message - Annex A.3.2.1</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Status">
          <xs:simpleType>
            <xs:restriction base="xs:int">
              <xs:enumeration value="0">
                <xs:annotation>
                  <xs:documentation>All Parameter changes have been validated and
                     applied</xs:documentation>
                </xs:annotation>
              </xs:enumeration>
              <xs:enumeration value="1">
                <xs:annotation>
                  <xs:documentation>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)</xs:documentation>
                </xs:annotation>
              </xs:enumeration>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- GetParameterValues -->
  <xs:element name="GetParameterValues">
    <xs:annotation>
      <xs:documentation>GetParameterValues message - Annex A.3.2.2</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ParameterNames" type="tns:ParameterNames"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- GetParameterValuesResponse -->
  <xs:element name="GetParameterValuesResponse">
    <xs:annotation>
      <xs:documentation>GetParameterValuesResponse message - Annex A.3.2.2</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ParameterList" type="tns:ParameterValueList"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- GetParameterNames -->
  <xs:element name="GetParameterNames">
    <xs:annotation>
      <xs:documentation>GetParameterNames message - Annex A.3.2.3</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ParameterPath" nillable="true">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:maxLength value="256"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="NextLevel" type="xs:boolean"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- GetParameterNamesResponse -->
  <xs:element name="GetParameterNamesResponse">
    <xs:annotation>
      <xs:documentation>GetParameterNamesResponse message - Annex A.3.2.3</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ParameterList" type="tns:ParameterInfoList"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- SetParameterAttributes -->
  <xs:element name="SetParameterAttributes">
    <xs:annotation>
      <xs:documentation>SetParameterAttributes message - Annex A.3.2.4</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ParameterList" type="tns:SetParameterAttributesList"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- SetParameterAttributesResponse -->
  <xs:element name="SetParameterAttributesResponse">
    <xs:annotation>
      <xs:documentation>SetParameterAttributesResponse message - Annex A.3.2.4</xs:documentation>
    </xs:annotation>
    <xs:complexType/>
  </xs:element>

  <!-- GetParameterAttributes -->
  <xs:element name="GetParameterAttributes">
    <xs:annotation>
      <xs:documentation>GetParameterAttributes message - Annex A.3.2.5</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ParameterNames" type="tns:ParameterNames"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- GetParameterAttributesResponse -->
  <xs:element name="GetParameterAttributesResponse">
    <xs:annotation>
      <xs:documentation>GetParameterAttributesResponse message - Annex A.3.2.5</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ParameterList" type="tns:ParameterAttributeList"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- AddObject -->
  <xs:element name="AddObject">
    <xs:annotation>
      <xs:documentation>AddObject message - Annex A.3.2.6</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ObjectName" type="tns:ObjectNameType"/>
        <xs:element name="ParameterKey" type="tns:ParameterKeyType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- AddObjectResponse -->
  <xs:element name="AddObjectResponse">
    <xs:annotation>
      <xs:documentation>AddObjectResponse message - Annex A.3.2.6</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="InstanceNumber">
          <xs:simpleType>
            <xs:restriction base="xs:unsignedInt">
              <xs:minInclusive value="1"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="Status">
          <xs:simpleType>
            <xs:restriction base="xs:int">
              <xs:enumeration value="0">
                <xs:annotation>
                  <xs:documentation>The object has been created</xs:documentation>
                </xs:annotation>
              </xs:enumeration>
              <xs:enumeration value="1">
                <xs:annotation>
                  <xs:documentation>The object creation has been validated and committed, but not
                      yet applied</xs:documentation>
                </xs:annotation>
              </xs:enumeration>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

```
<!-- DeleteObject -->
<xs:element name="DeleteObject">
  <xs:annotation>
    <xs:documentation>DeleteObject message - Annex A.3.2.7</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ObjectName" type="tns:ObjectNameType"/>
      <xs:element name="ParameterKey" type="tns:ParameterKeyType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- DeleteObjectResponse -->
<xs:element name="DeleteObjectResponse">
  <xs:annotation>
    <xs:documentation>DeleteObjectResponse message - Annex A.3.2.7</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Status">
        <xs:simpleType>
          <xs:restriction base="xs:int">
            <xs:enumeration value="0">
              <xs:annotation>
                <xs:documentation>The object has been deleted</xs:documentation>
              </xs:annotation>
            </xs:enumeration>
            <xs:enumeration value="1">
              <xs:annotation>
                <xs:documentation>The object deletion has been validated and committed, but not
                    yet applied</xs:documentation>
              </xs:annotation>
            </xs:enumeration>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- Download -->
<xs:element name="Download">
  <xs:annotation>
    <xs:documentation>Download message - Annex A.3.2.8</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="CommandKey" type="tns:CommandKeyType"/>
      <xs:element name="FileType">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="64"/>
            <xs:pattern value="1 Firmware Upgrade Image"/>
            <xs:pattern value="2 Web Content"/>
            <xs:pattern value="3 Vendor Configuration File"/>
            <xs:pattern value="X [0-9A-F]{6} .*"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="URL">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="256"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Username">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="256"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Password">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="256"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="FileSize" type="xs:unsignedInt"/>
      <xs:element name="TargetFileName">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="256"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="DelaySeconds" type="xs:unsignedInt"/>
      <xs:element name="SuccessURL">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="256"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="FailureURL">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="256"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- DownloadResponse -->
<xs:element name="DownloadResponse">
  <xs:annotation>
    <xs:documentation>DownloadResponse message - Annex A.3.2.8</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Status">
        <xs:simpleType>
          <xs:restriction base="xs:int">
            <xs:enumeration value="0">
              <xs:annotation>
                <xs:documentation>Download has completed and been applied</xs:documentation>
              </xs:annotation>
            </xs:enumeration>
            <xs:enumeration value="1">
              <xs:annotation>
                <xs:documentation>Download has not yet been completed and
                    applied</xs:documentation>
              </xs:annotation>
            </xs:enumeration>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="StartTime" type="xs:dateTime"/>
      <xs:element name="CompleteTime" type="xs:dateTime"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- Reboot -->
<xs:element name="Reboot">
  <xs:annotation>
    <xs:documentation>Reboot message - Annex A.3.2.9</xs:documentation>
```

```
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="CommandKey" type="tns:CommandKeyType"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <!-- RebootResponse -->
    <xs:element name="RebootResponse">
      <xs:annotation>
        <xs:documentation>RebootResponse message - Annex A.3.2.9</xs:documentation>
      </xs:annotation>
      <xs:complexType/>
    </xs:element>


    <!--
         Optional CPE messages - Annex A.4.1
    -->
    <!-- GetQueuedTransfers -->
    <xs:element name="GetQueuedTransfers">
      <xs:annotation>
        <xs:documentation>GetQueuedTransfers message - Annex A.4.1.1</xs:documentation>
      </xs:annotation>
      <xs:complexType/>
    </xs:element>

    <!-- GetQueuedTransfersResponse -->
    <xs:element name="GetQueuedTransfersResponse">
      <xs:annotation>
        <xs:documentation>GetQueuedTransfersResponse message - Annex A.4.1.1</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="TransferList" type="tns:TransferList"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <!-- ScheduleInform -->
    <xs:element name="ScheduleInform">
      <xs:annotation>
        <xs:documentation>ScheduleInform message - Annex A.4.1.2</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="DelaySeconds" type="xs:unsignedInt"/>
          <xs:element name="CommandKey" type="tns:CommandKeyType"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <!-- ScheduleInformResponse -->
    <xs:element name="ScheduleInformResponse">
      <xs:annotation>
        <xs:documentation>ScheduleInformResponse message - Annex A.4.1.2</xs:documentation>
      </xs:annotation>
      <xs:complexType/>
    </xs:element>

    <!-- SetVouchers -->
    <xs:element name="SetVouchers">
      <xs:annotation>
        <xs:documentation>SetVouchers message - Annex A.4.1.3</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="VoucherList" type="tns:VoucherList"/>
        </xs:sequence>
      </xs:complexType>
```

```
  </xs:element>

  <!-- SetVouchersResponse -->
  <xs:element name="SetVouchersResponse">
    <xs:annotation>
      <xs:documentation>SetVouchersResponse message - Annex A.4.1.3</xs:documentation>
    </xs:annotation>
    <xs:complexType/>
  </xs:element>

  <!-- GetOptions -->
  <xs:element name="GetOptions">
    <xs:annotation>
      <xs:documentation>GetOptions message - Annex A.4.1.4</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="OptionName">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:maxLength value="64"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- GetOptionsResponse -->
  <xs:element name="GetOptionsResponse">
    <xs:annotation>
      <xs:documentation>GetOptionsResponse message - Annex A.4.1.4</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="OptionList" type="tns:OptionList"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- Upload -->
  <xs:element name="Upload">
    <xs:annotation>
      <xs:documentation>Upload message - Annex A.4.1.5</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="CommandKey" type="tns:CommandKeyType"/>
        <xs:element name="FileType">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:maxLength value="64"/>
              <xs:pattern value="1 Vendor Configuration File"/>
              <xs:pattern value="2 Vendor Log File"/>
              <xs:pattern value="X [0-9A-F]{6} .*"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="URL">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:maxLength value="256"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="Username">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:maxLength value="256"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="Password">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:maxLength value="256"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="DelaySeconds" type="xs:unsignedInt"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- UploadResponse -->
  <xs:element name="UploadResponse">
    <xs:annotation>
      <xs:documentation>UploadResponse message - Annex A.4.1.5</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Status">
          <xs:simpleType>
            <xs:restriction base="xs:int">
              <xs:enumeration value="0">
                <xs:annotation>
                  <xs:documentation>Upload has been completed</xs:documentation>
                </xs:annotation>
              </xs:enumeration>
              <xs:enumeration value="1">
                <xs:annotation>
                  <xs:documentation>Upload has not yet completed</xs:documentation>
                </xs:annotation>
              </xs:enumeration>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="StartTime" type="xs:dateTime"/>
        <xs:element name="CompleteTime" type="xs:dateTime"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- FactoryReset -->
  <xs:element name="FactoryReset">
    <xs:annotation>
      <xs:documentation>FactoryReset message - Annex A.4.1.6</xs:documentation>
    </xs:annotation>
    <xs:complexType/>
  </xs:element>

  <!-- FactoryResetResponse -->
  <xs:element name="FactoryResetResponse">
    <xs:annotation>
      <xs:documentation>FactoryResetResponse message - Annex A.4.1.6</xs:documentation>
    </xs:annotation>
    <xs:complexType/>
  </xs:element>


  <!--
       ACS messages - Annex A.3.3
  -->
  <!-- Inform -->
  <xs:element name="Inform">
    <xs:annotation>
      <xs:documentation>Inform message - Annex A.3.3.1</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="DeviceId" type="tns:DeviceIdStruct"/>
        <xs:element name="Event" type="tns:EventList"/>
        <xs:element name="MaxEnvelopes" type="xs:unsignedInt"/>
        <xs:element name="CurrentTime" type="xs:dateTime"/>
        <xs:element name="RetryCount" type="xs:unsignedInt"/>
        <xs:element name="ParameterList" type="tns:ParameterValueList"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- InformResponse -->
  <xs:element name="InformResponse">
    <xs:annotation>
      <xs:documentation>InformResponse message - Annex A.3.3.1</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="MaxEnvelopes" type="xs:unsignedInt"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- TransferComplete -->
  <xs:element name="TransferComplete">
    <xs:annotation>
      <xs:documentation>TransferComplete message - Annex A.3.3.2</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="CommandKey" type="tns:CommandKeyType"/>
        <xs:element name="FaultStruct" type="tns:FaultStruct"/>
        <xs:element name="StartTime" type="xs:dateTime"/>
        <xs:element name="CompleteTime" type="xs:dateTime"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- TransferCompleteResponse -->
  <xs:element name="TransferCompleteResponse">
    <xs:annotation>
      <xs:documentation>TransferCompleteResponse message - Annex A.3.3.2</xs:documentation>
    </xs:annotation>
    <xs:complexType/>
  </xs:element>


  <!--
       Optional ACS messages - Annex A.4.2
  -->
  <!-- Kicked -->
  <xs:element name="Kicked">
    <xs:annotation>
      <xs:documentation>Kicked message - Annex A.4.2.1</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Command">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:maxLength value="32"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="Referer">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:maxLength value="64"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="Arg">
          <xs:simpleType>
            <xs:restriction base="xs:string">
```

```
              <xs:maxLength value="256"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="Next">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:maxLength value="1024"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- KickedResponse -->
  <xs:element name="KickedResponse">
    <xs:annotation>
      <xs:documentation>KickedResponse message - Annex A.4.2.1</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="NextURL">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:maxLength value="1024"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- RequestDownload -->
  <xs:element name="RequestDownload">
    <xs:annotation>
      <xs:documentation>RequestDownload message - Annex A.4.2.2</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="FileType">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:maxLength value="64"/>
              <xs:pattern value="1 Firmware Upgrade Image"/>
              <xs:pattern value="2 Web Content"/>
              <xs:pattern value="3 Vendor Configuration File"/>
              <xs:pattern value="X [0-9A-F]{6} .*"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="FileTypeArg" type="tns:FileTypeArg"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- RequestDownloadResponse -->
  <xs:element name="RequestDownloadResponse">
    <xs:annotation>
      <xs:documentation>RequestDownloadResponse message - Annex A.4.2.2</xs:documentation>
    </xs:annotation>
    <xs:complexType/>
  </xs:element>

</xs:schema>
```

Annex B. Removed

Annex Removed.

Annex C. Signed Vouchers

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 [15]. 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 [15]. 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 62. An Option MAY contain additional XML elements specific to the particular Option. An example Option specification structure is shown in Figure 5. An example of an entire signed Voucher is shown in Figure 6. In this example, two separate Options are enabled in the same Voucher.

Table 62 – 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 63. 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"
`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 [13].

Table 63 – 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.

Name	Type	Description
`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 [9].
`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 5 – 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 6 – 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

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.

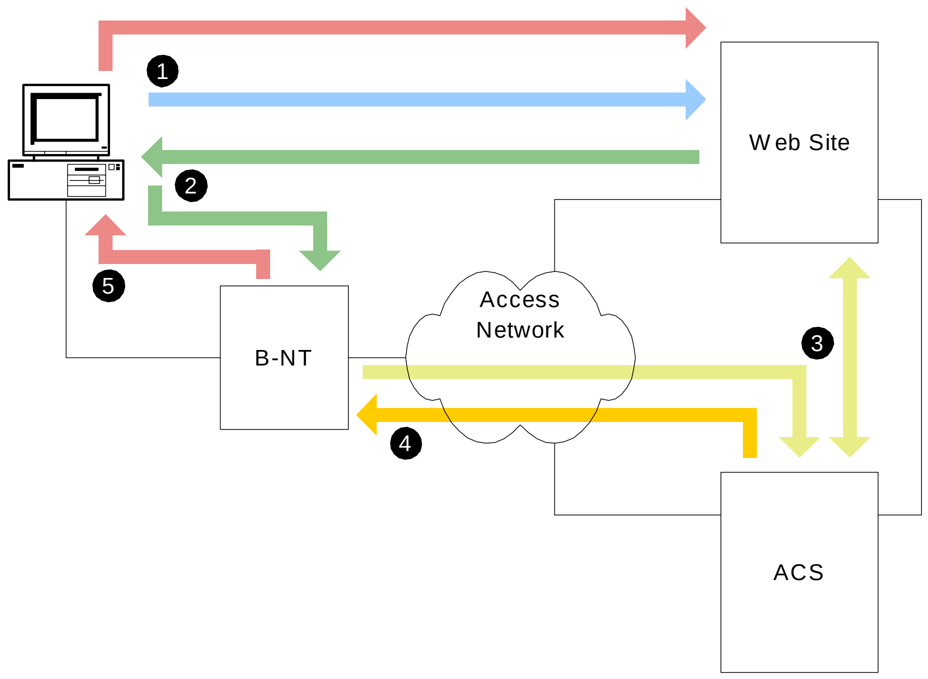
2. The web site redirects the browser to a specific URL accessible only from the CPE's private-network (LAN) interface through which the browser "kicks" the CPE, providing the CPE via CGI arguments with information it needs to follow the subsequent steps (see section D.4).
3. The CPE notifies the ACS 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 7 shows the sequence of events associated with this mechanism. The numbers shown correspond to the step numbers above.

Figure 7 – 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 64.

Table 64 – 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 64. 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

E.1 Introduction

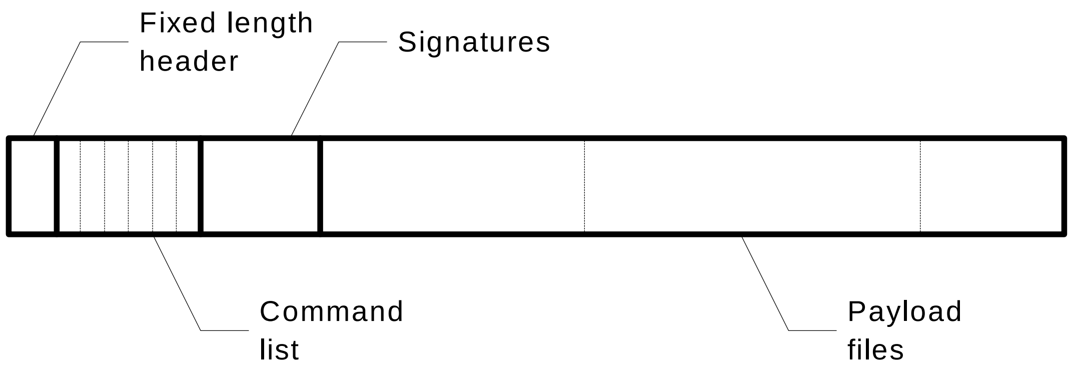
This document specifies a signed package format that MAY be used to securely download files into a recipient device. The format allows one or more files to be encapsulated within a single signed package. The package format allows the recipient to authenticate the source, and contains instructions for 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 8.

Figure 8 – Signed package format



A general description of each of the signed package format components is given in Table 65.

Table 65 – 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.
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 66.

Table 66 – 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 2^{16}.
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 67.

Table 67 – 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 68.

Table 68 – 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 69.

Table 69 – 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.
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 70.

Table 70 – 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 71.

Table 71 – 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.

Field	Type	Description
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.

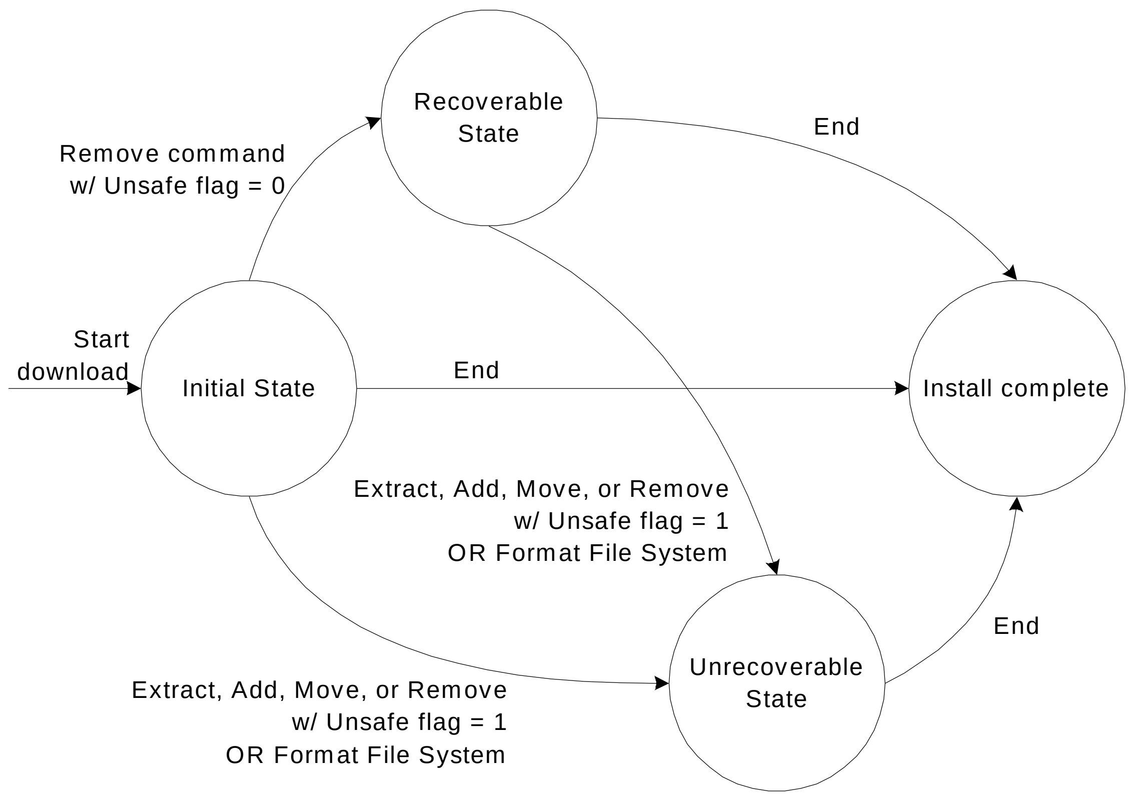
Each timeout command includes information in the Value portion of the command. The format of this information is defined in Table 72.

Table 72 – 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 9. 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 9 – 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 73.

Table 73 – Timeout command definitions

Command	Description
Initial Timeout	This command sets the download timeout used during the Initial State as shown in Figure 9. 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 9. 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 9. 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 9. 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 74.

Table 74 – 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 75.

Table 75 – 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 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 [9]. 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 76.

Table 76 – 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 77.

Table 77 – 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 [9]. 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 [16].

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 [20]. 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 [24].

- 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 [13]. 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:1 profile as defined in [24].

A Gateway conforming to this Annex MUST inspect all DHCP 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 V-I Vendor-Specific Information DHCP Option (option number 125, as defined in [22]) that includes the Device Identity information, as defined in section F.2.5. The DHCP requests for which this requirement applies are DHCPDISCOVER, DHCPREQUEST, and DHCPINFORM.

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 V-I Vendor-Specific Information DHCP Option (option number 125, as defined in [22]), as defined in section F.2.5. The DHCP responses for which this requirement applies are DHCPOFFER and DHCPACK.
- On successful completion of the DHCP exchange (following the DHCPACK), 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 [24]) 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 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 InternetGatewayDevice.ManagementServer.ManageableDeviceNumberOfEntries 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 [13].

A Device conforming to this Annex MUST do the following:

- In DHCP requests, the Device MUST include a V-I Vendor-Specific Information DHCP Option (option number 125, as defined in [22]) that includes its Device Identity information, as defined in section F.2.5. The DHCP requests for which this requirement applies are DHCPDISCOVER, DHCPREQUEST, and DHCPINFORM.
- If the DHCPACK message includes the Gateway Identity carried in the V-I Vendor-Specific Information DHCP Option (option number 125, as defined in [22]), as defined in section F.2.5, the Device MUST record the received value in the GatewayInfo data object defined in [13]. All of the following values MUST be recorded:

  Device.GatewayInfo.ManufacturerOUI

  Device.GatewayInfo.SerialNumber

  Device.GatewayInfo.ProductClass

  The DHCP responses for which this requirement applies are DHCPOFFER and DHCPACK.
- 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 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 [24]). The ACS would determine the current contents of the Manageable-Device 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 ManageableDeviceNumberOfEntries parameter).

F.2.4 Device-Gateway Association Flows

Figure 10 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.

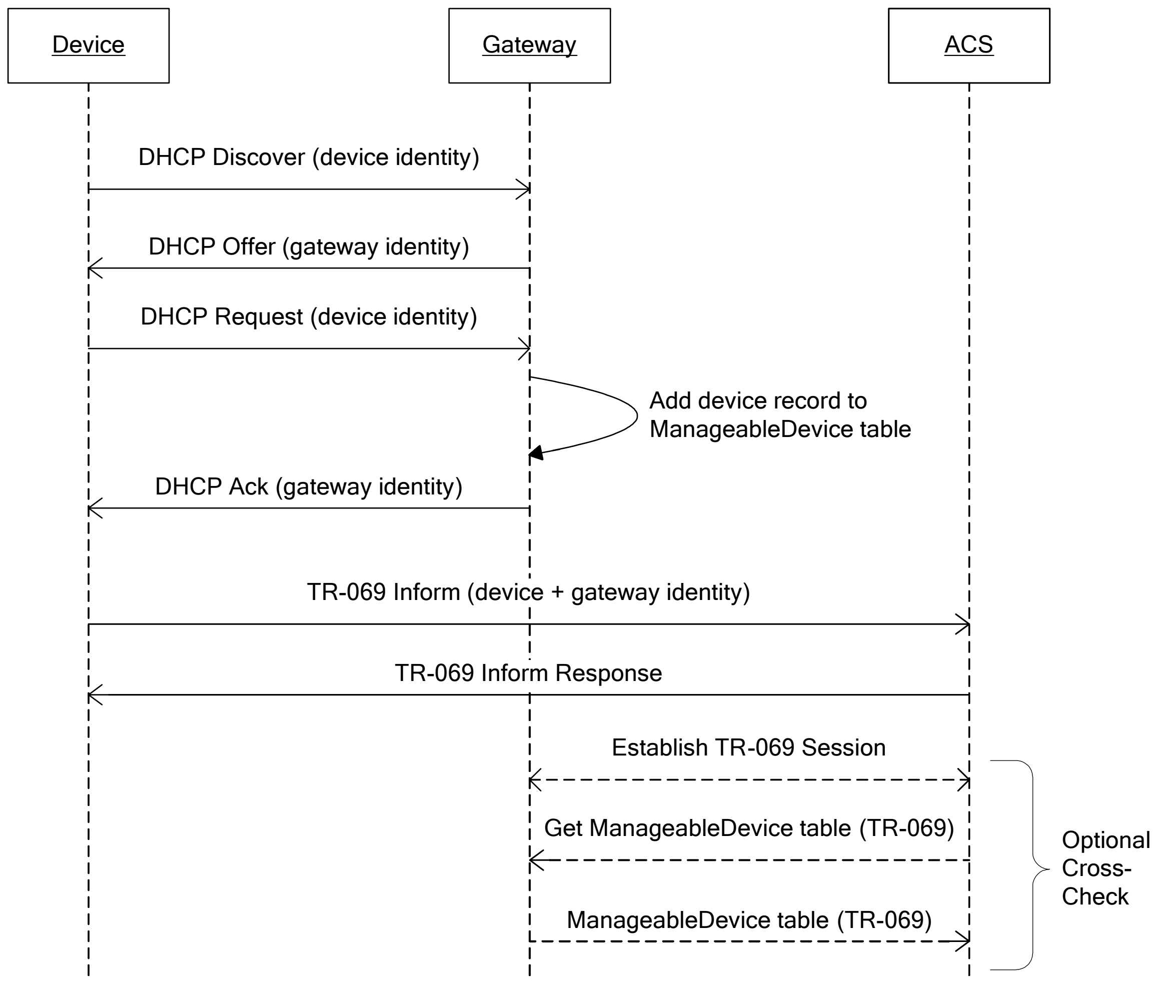

Device
Gateway
ACS
DHCP Discover (device identity)
DHCP Offer (gateway identity)
DHCP Request (device identity)
Add device record to ManageableDevice table
DHCP Ack (gateway identity)
TR-069 Inform (device + gateway identity)
TR-069 Inform Response
Establish TR-069 Session
Get ManageableDevice table (TR-069)
ManageableDevice table (TR-069)
Optional Cross-Check

Figure 10 – 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 11.

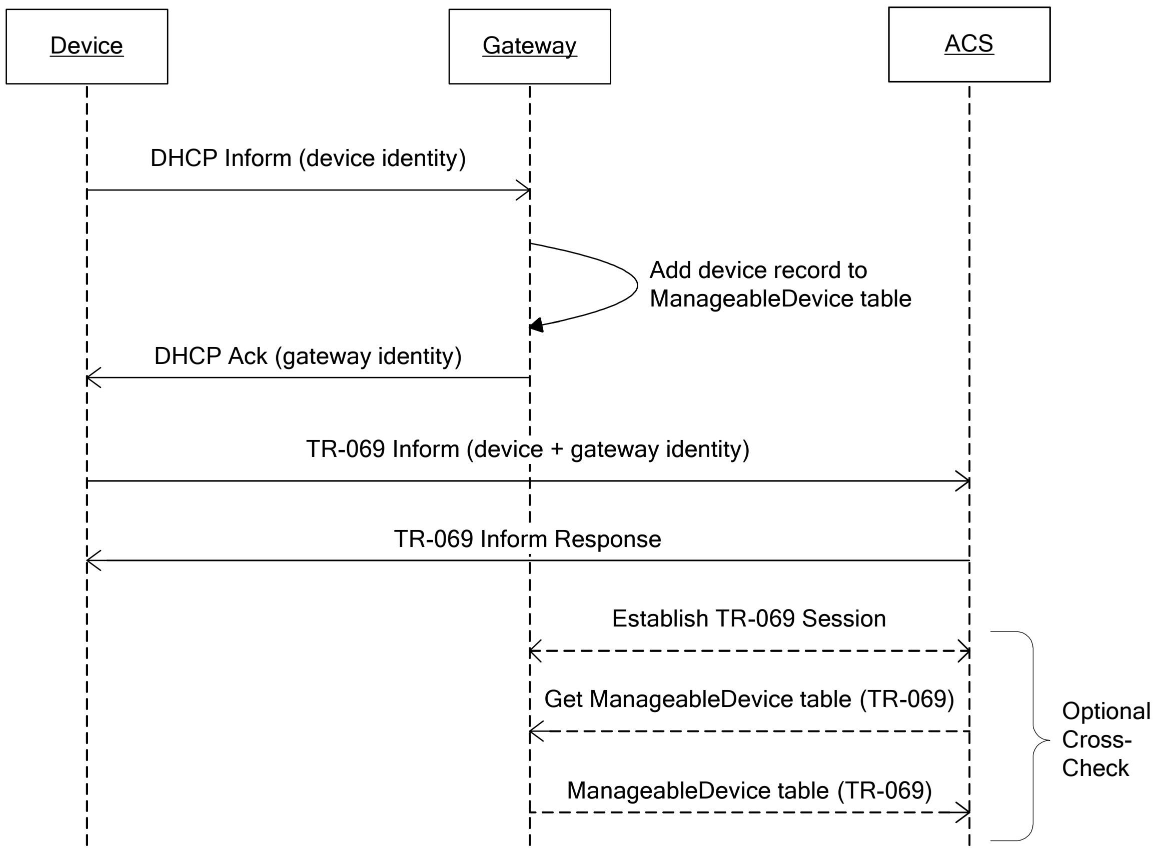

Device
Gateway
ACS
DHCP Inform (device identity)
Add device record to ManageableDevice table
DHCP Ack (gateway identity)
TR-069 Inform (device + gateway identity)
TR-069 Inform Response
Establish TR-069 Session
Get ManageableDevice table (TR-069)
ManageableDevice table (TR-069)
Optional Cross-Check

Figure 11 – 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 V-I Vendor-Specific Information DHCP Option, which is option number 125, as defined in [22]. This DHCP option is 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 V-I 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 [18]).

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 [22]. 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 78.

Table 78 – Encapsulated Vendor-Specific Option-Data fields

Encapsulated Option	Sub-Option Code	Source Entity	Source Parameter[11]
DeviceManufacturerOUI	1	Device	Device.Info.ManufacturerOUI[12]
DeviceSerialNumber	2	Device	Device.Info.SerialNumber[12]
DeviceProductClass	3	Device	Device.Info.ProductClass[12]
GatewayManufacturerOUI	4	Gateway	InternetGatewayDevice.DeviceInfo.ManufacturerOUI[13]
GatewaySerialNumber	5	Gateway	InternetGatewayDevice.DeviceInfo.SerialNumber[13]
GatewayProductClass	6	Gateway	InternetGatewayDevice.DeviceInfo.ProductClass[13]

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 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.

[11] The value of the corresponding Sub-Option Data element is obtained from the specified parameter value.
[12] As defined in [13].
[13] As defined in [24].

Annex G. Connection Request via NAT Gateway

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 [21].

The use of STUN for this purpose requires that a new UDP-based Connection Request mechanism be defined to augment the existing TCP-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 [24] if the CPE is an Internet Gateway Device, or as defined in [13] 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 [21] 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 UDPConnectionRequest-Address 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 [21]. 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 [21]. 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 [21], 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 STUN-MinimumKeepAlivePeriod 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.[14]

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.[15]

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 STUN-MinimumKeepAlivePeriod parameter in the ManagementServer object.

The CPE MUST send these Binding Requests at least as frequently as is specified by the STUNMaximum-KeepAlivePeriod 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 [21] 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).

[14] 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.

[15] These attributes are primarily intended to allow discovery of the type of NAT in use, which is not required for this Annex.

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 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 STUNMinimumKeepAlivePeriod 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.[16] 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 79 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 79 – 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:[17]
		“broadband-forum.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.

[16] 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.

[17] 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.

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.

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 STUNMinimumKeep-AlivePeriod 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 [21] 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 UDPConnectionRequestAddress-NotificationLimit, 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 TCP-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 in [5] 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 Device.ManagementServer.ConnectionRequestUsername.

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 Device.ManagementServer.ConnectionRequestPassword.

Whenever a CPE receives and successfully authenticates and validates a UDP Connection Request, it MUST follow the same requirements as for a TCP-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 [21] 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 [17]), 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 [21], 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 [21]. 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 [21].

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 [21] 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 TCP-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 UDPConnection-RequestAddress changes, or by enabling Active Notification on this parameter as well). Whenever this parameter is false, the ACS MAY use TCP-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 [5] 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 as defined in [5].
- 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 [12]. 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 [10]. The ACS MAY set this to the value of Device.ManagementServer.UDPConnectionRequestAddress, 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 [23]. 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 Device.ManagementServer.Connection-RequestUsername 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) [19], where: • Key is the value of the parameter Device.ManagementServer.Connection-RequestPassword 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 12 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.

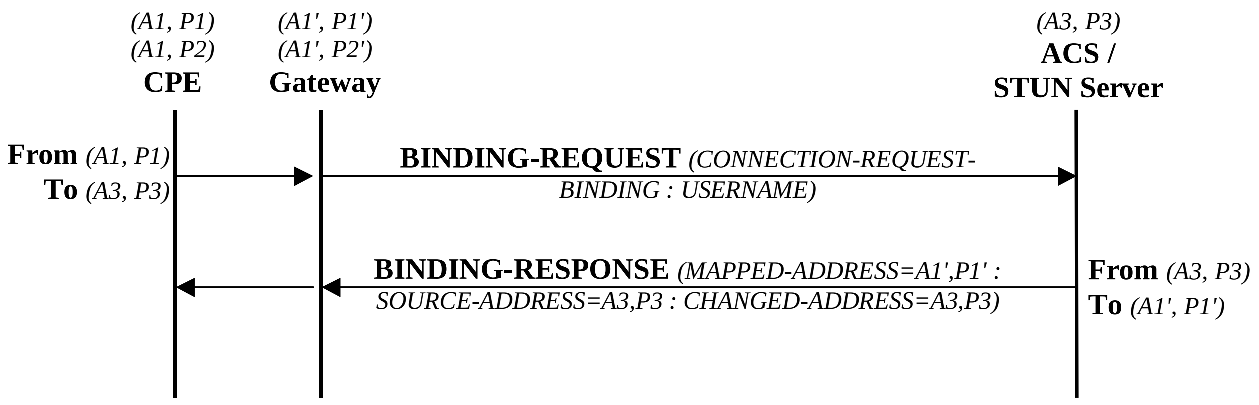

(A1, P1)
(A1, P2)
CPE
(A1', P1')
(A1', P2')
Gateway
(A3, P3)
ACS / STUN Server
From (A1, P1)
To (A3, P3)
BINDING-REQUEST (CONNECTION-REQUEST-BINDING : USERNAME)
BINDING-RESPONSE (MAPPED-ADDRESS=A1',P1' : SOURCE-ADDRESS=A3,P3 : CHANGED-ADDRESS=A3,P3)
From (A3, P3)
To (A1', P1')

Figure 12 – Binding discovery / maintenance from the primary source port

Figure 13 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.

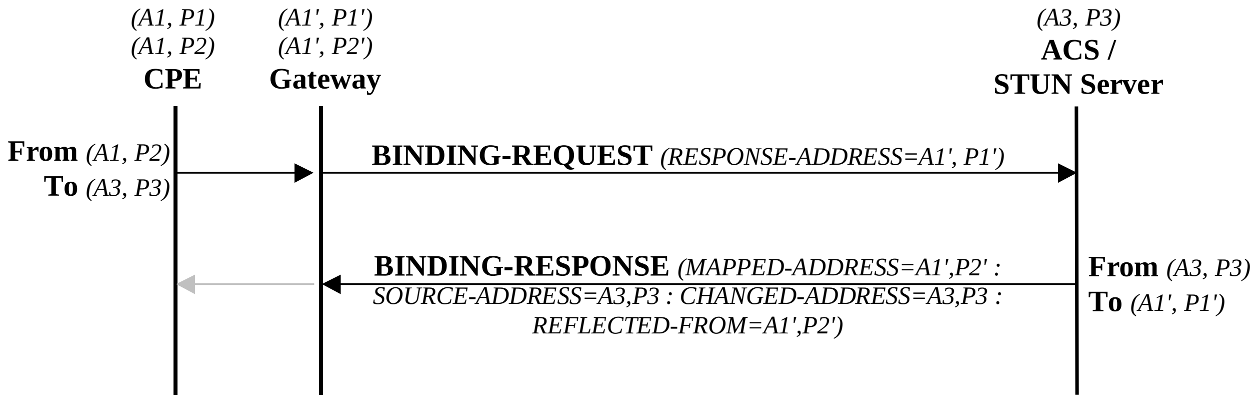
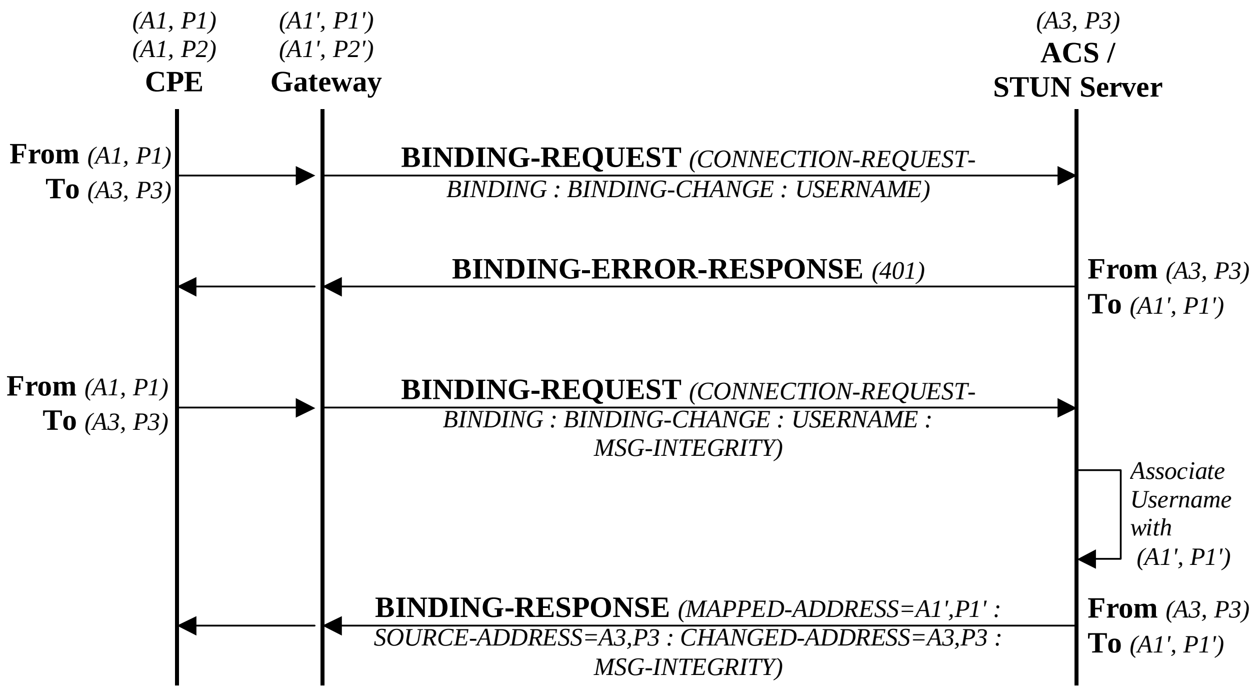

(A1, P1)
(A1, P2)
CPE
(A1', P1')
(A1', P2')
Gateway
(A3, P3)
ACS / STUN Server
From (A1, P2)
To (A3, P3)
BINDING-REQUEST (RESPONSE-ADDRESS=A1', P1')
BINDING-RESPONSE (MAPPED-ADDRESS=A1',P2' : SOURCE-ADDRESS=A3,P3 : CHANGED-ADDRESS=A3,P3 : REFLECTED-FROM=A1',P2')
From (A3, P3)
To (A1', P1')

Figure 13 – Binding Request from secondary source port for binding timeout discovery

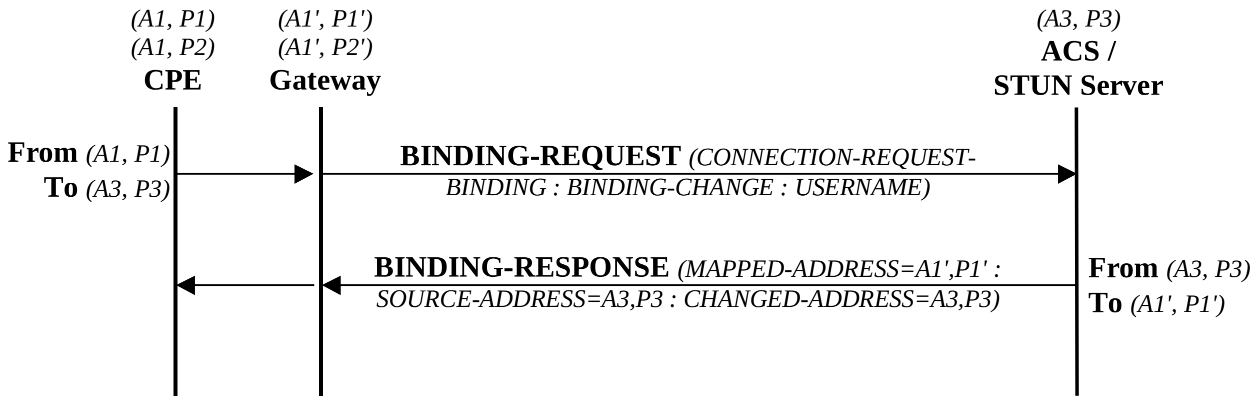
Figure 14 shows a Binding Change notification where the STUN Server has chosen to make use of the STUN-based approach (see section G.2.2.2.1), and therefore authenticates the Binding Request prior to storing the information associating the Username with the current binding address and port.



(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 14 – Binding change notification authenticated by the ACS

Figure 15 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 15 – Binding change notification *not* authenticated by the ACS

Figure 16 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.

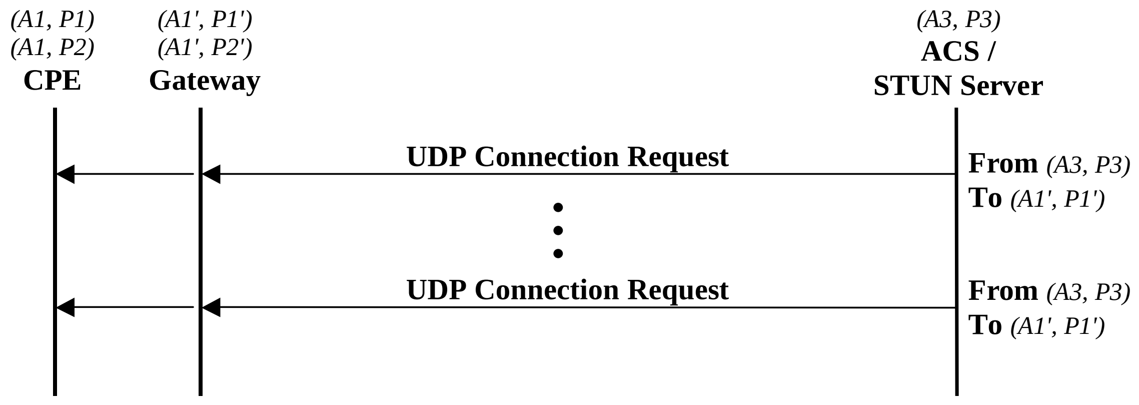


Figure 16 – UDP Connection Request

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 [21] 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.