TECHNICAL REPORT

DSL Forum
TR-069

CPE WAN Management Protocol

May 2004

Produced by:
DSLHome-Technical Working Group

Editors:
Jeff Bernstein, 2Wire
Tim Spets, Westell

Working Group Co-Chairs:
Greg Bathrick, Texas Instruments
George Pitsoulakis, Westell

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.

Contents

1 Introduction

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.

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 includes specific provisioning parameters and a general mechanism for adding vendor-specific provisioning capabilities as needed.

The provisioning mechanism allows CPE provisioning at the time of initial connection to the broadband access network, and the ability to re-provision 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 Appendix 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. The protocol defines a common set of such parameters, and provides a standard syntax for vendors to define additional non-standard parameters that an ACS can monitor. It also defines a set of conditions under which a CPE should actively notify the ACS of changes.

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 connectivity or service issues. The protocol defines a common set of such parameters and a general mechanism for adding vendor-specific diagnostic capabilities.

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 Appendix D.

1.2 Positioning in the Auto-Configuration Architecture

TR-046 [2] describes the overall framework for B-NT auto-configuration. This process consists of three sequential stages, each of which is focused on a specific aspect of the overall B-NT auto-configuration process.

The procedures for the first two stages of B-NT auto-configuration are specified in TR-062 [3] and TR-044 [4]. These define the ATM layer and IP layer auto-configuration procedures, respectively, used to initiate basic broadband connectivity.

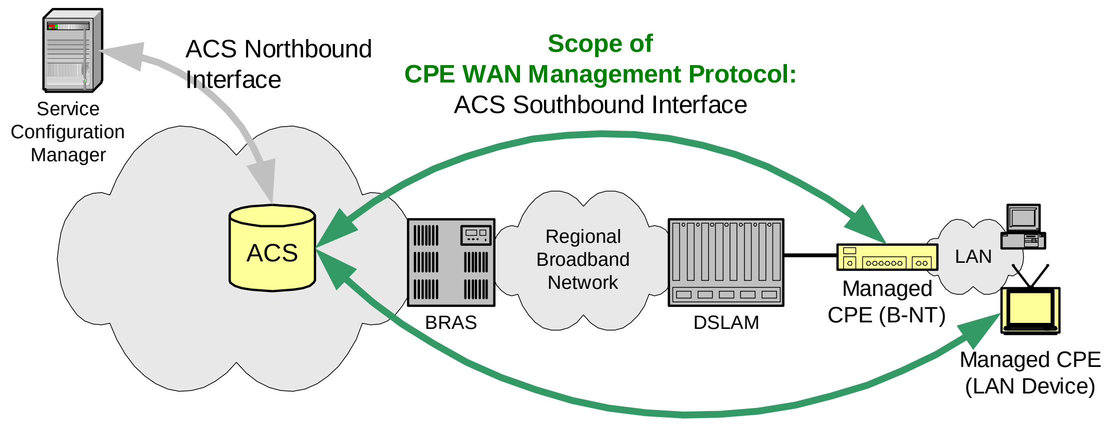
The third stage of auto-configuration defined in TR-046 covers "auto-configured complex services." In the case of a B-NT, the CPE WAN Management Protocol relates primarily to this third stage. Specifically, the CPE WAN Management Protocol is proposed as the protocol to be used on the ACS-Southbound Interface between an Auto-Configuration Server (ACS), and a B-NT as shown in Figure 1.

Note—in the case of a B-NT, contrary to the nested model of TR-046, this protocol also allows configuration of ATM Layer parameters if an alternative auto-configuration protocol is not in use, e.g., as defined in TR-062. However, if an alternative is in use then configuration of the ATM Layer parameters by this protocol is disabled.

In addition to configuration, the protocol provides a means of extracting diagnostic and performance monitoring data from the ATM layer and the DSL modem. Again this is contrary to the nested model described in TR-046, but provides an alternative means of accessing information that can already be obtained through existing management protocols, i.e., ILMI and the EOC of the DSL link. The provision of more advanced diagnostic and performance monitoring functionality via this protocol is a subject for further study.

While the CPE WAN Management Protocol is targeted at management of B-NTs, this protocol may be used to manage other types of CPE as well, including stand-alone routers and LAN-side client devices, as also shown in Figure 1. Unless otherwise indicated, the CPE WAN Management Protocol as defined in this specification applies to any such managed device. Portions of this specification that apply only to a B-NT are explicitly indicated in the text. This specification includes a complete definition of the CPE parameter model for a B-NT. The corresponding parameter model for other specific device types is beyond the scope of this specification.

Figure 1 – Positioning in the Auto-Configuration 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 flexible support for various business models for distributing and managing CPE, including:

- CPE provided and managed by the network provider.
- CPE purchased in retail with pre-registration to associate the specific CPE with a service provider and customer account (a mobile-phone like model)
- CPE purchased in retail with post-installation user registration with a service provider.

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 ACS servers to serve a population of CPE, which may be associated with one or more service providers.

- Optimize the use of connections that are established to minimize connection overhead by allowing multiple bi-directional transactions to occur over a single connection.

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

- Different CPE may have differing capability levels, implementing different subsets of optional functionality. 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 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:

- In the case of a B-NT, prior to use of the CPE WAN Management Protocol, initial B-NT auto-configuration as defined in TR-062 [3] and TR-044 [4] has been completed and a connection has been established to a WAN from which an ACS is accessible.
- 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. An ACS can hand off a CPE to another ACS only by explicitly altering the ACS contact and authentication information. (Note: a collection of ACS servers 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.

[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 bridge may connect using IPoE with DHCP for address allocation, or may connect using PPPoE.

B-NT	A broadband access CPE device capable of being managed by an ACS.
CPE	Customer Premise Equipment. A DSL B-NT is one form of broadband CPE.
Internet Gateway Device	A CPE device that is either a B-NT or a broadband router.
Option	An optional CPE capability that may only be enabled or disabled using a digitally signed Voucher.
RPC	Remote procedure call.
Parameter	A name-value pair representing a manageable CPE parameter made accessible to an ACS for reading and/or writing.
Session	A contiguous sequence of transactions between a CPE and an ACS.
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.

Figure 2 – Protocol stack

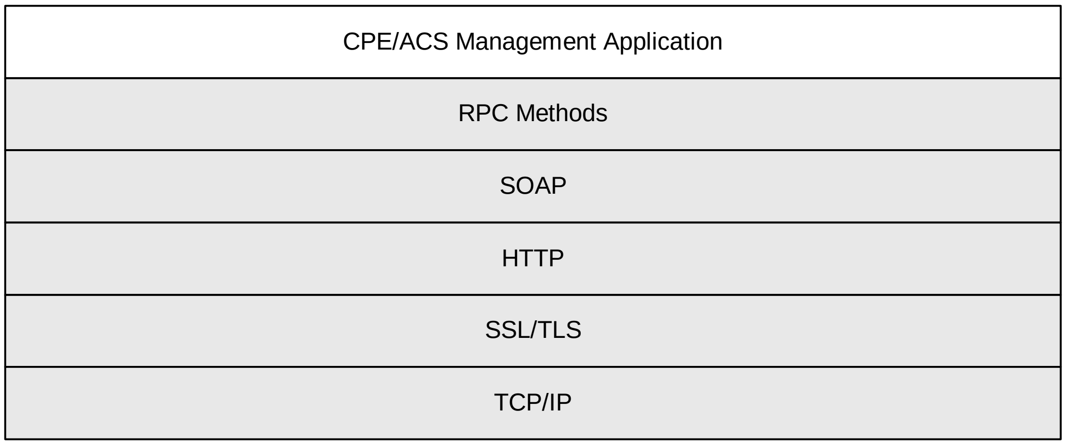


Table 1 – Protocol layer summary

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 Appendix A. This includes the definition of the CPE Parameters accessible by an ACS via the Parameter-related RPC Methods. The specific Parameters defined for an Internet Gateway Device are specified in Appendix B.
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]. Use of SSL/TLS is RECOMMENDED but is not required.
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 authentication based on shared secrets.

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

2.2.1 Security Initialization Models

Initialization of the security mechanisms is described in the context of various business models for CPE distribution. Three models are considered:

- Distribution of CPE by the service provider associated with the ACS.
- Retail distribution of the CPE, where association of the CPE with the service provider and customer is done at the time of purchase.
- Retail distribution where no pre-association with the CPE is done.

In the first two cases, the specific identity of the CPE can be known to the ACS before the CPE is first used. In these cases, the following mechanisms may be used:

Authentication of	Type used	Description
ACS	Shared secret	Shared secret must be pre-loaded into CPE before the first use of the CPE.
	Certificate	Discovery of the ACS URL as described in section 3.1 uniquely identifies the identity of the ACS for the purpose of certificate validation.
CPE	Shared secret	Shared secret must be provided to the ACS before the first use of the CPE.
	Certificate	The CPE may use online certificate enrollment with the CA associated with the ACS. The CPE must be provided with the information needed to contact this CA.

In the latter case of retail distribution of the CPE, there is no possibility of pre-association of the CPE with a particular ACS. The following table presents possible approaches to accommodating this case, but does not attempt to mandate a specific approach:

Authentication of	Type used	Description
ACS	Shared secret	Not appropriate for this case.
	Certificate	Discovery of the ACS URL as described in section 3.1 uniquely identifies the identity of the ACS for the purpose of certificate validation.
CPE	Shared secret	Possible alternatives, outside the scope of this specification: • Establish a common server for secure distribution of CPE shared secrets among multiple service providers. • Initial CPE to ACS connection of an unrecognized CPE could be allowed without authentication. ACS would then set the shared secret Parameter for subsequent access. Care in ACS implementation would be required to prevent denial of service attacks.
	Certificate	The CPE may use online certificate enrollment with the CA associated with the ACS. The CPE must be provided with the information needed to contact this CA, which could be incorporated into the discovery process.

2.3 Architectural Components

2.3.1 Parameters

The RPC Method Specification (see Appendix A) defines a generic mechanism by which an ACS can read or write Parameters to configure a CPE and monitor CPE status and statistics. The particular list of defined Parameters for an Internet Gateway Device is specified in Appendix B.

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 Appendix B).

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. 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).

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 Appendix 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 or, optionally, HTTPS, FTP, 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 Appendix E.

2.3.3 CPE Initiated Notifications

The RPC Method Specification (see Appendix A) defines a mechanism that allows a CPE to notify 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, 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). The ACS must be aware of this event in order to establish incoming connections to the CPE.

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

2.3.4 Asynchronous ACS Initiated Notifications

An important aspect of service auto-configuration is the ability for the ACS to notify 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 Inform interval.

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.

3 Procedures and Requirements

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 [12]. 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 “dslforum.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 [12]. This list may include other vendor-specific options in addition to those listed here.

Table 2 – Encapsulated Vendor Specific Options

Encapsulated Option	Encapsulated Vendor-Specific Option number	Parameter[2]
URL of the ACS	1	InternetGatewayDevice.ManagementServer.URL
Provisioning code	2	InternetGatewayDevice.DeviceInfo.ProvisioningCode

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 ACS supports SSL. If an HTTPS URL is given, and the CPE that does not support SSL, it MAY attempt to use HTTP assuming the remainder of the URL is unchanged.

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 (InternetGatewayDevice.ManagementServer.URL). 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 may subsequently 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.

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. In the case of an Internet Gateway Device, this includes changes to the following (see A.3.3.1):

[2] As defined for an Internet Gateway Device.

    - IP address of the default broadband connection
    - Management IP address (associated with the Connection Request URL)
    - Provisioning code
    - Software version

- 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, only one notification is performed.

  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 effect 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.

The CPE SHOULD NOT maintain an open connection to the ACS when no more outstanding messages exist on the CPE or ACS.

3.2.2 ACS Connection Initiation

The ACS at any time request that the CPE initiate a connection to the ACS using the Connection Request notification 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 may not be able to access the CPE at all. In this case, only CPE connection initiation is possible.

The Connection Request notification mechanism is defined as follows:

- The Connection Request notification is an HTTP 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 notification 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 notification. Any data that might be contained SHOULD be ignored by the CPE.
- The CPE SHOULD use digest-authentication to authenticate the ACS before proceeding—the CPE SHOULD NOT initiate a connection to the ACS due to an unsuccessfully authenticated request. The shared-secret used to authenticate the ACS is available as a modifiable Parameter on the CPE.
- The CPE SHOULD restrict the number of Connection Request notifications it accepts during a given period of time in order to further reduce the possibility of a denial of service attack.
- The successful authentication of an HTTP Get to the designated port and URL causes the CPE to perform a fixed action: it establishes a session with the pre-determined ACS address (see section 3.1), and once connected, it sends an Inform message.
- If the CPE is already in a session with the ACS when it receives a Connection Request notification, it MUST NOT terminate that session prematurely as a result.

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 read the value of the InternetGatewayDevice.ManagementServer.ConnectionRequestURL Parameter. 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 (as described in Table 33), thus keeping the ACS up-to-date.

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:

- If SSL/TLS is supported, the REQUIRED versions are SSL 3.0 [10] or TLS 1.0 [11].
- If SSL/TLS is supported, support for encryption algorithms with key lengths greater than or equal to 128 bits SHOULD be supported.
- 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.
- If SSL/TLS is used, the ACS MAY accept a validated CPE-provided certificate to authenticate the CPE, but the ACS MUST allow the SSL/TLS connection to be established if the CPE does not provide a certificate.

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.

3.4.1 Encoding SOAP over HTTP

The encoding of SOAP over HTTP extends the basic HTTP profile for SOAP, described in [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.
- Each HTTP post or response may contain more than one SOAP envelope (within the negotiated limits). Each envelope may contain a SOAP request or response, independent from any other envelope.
- When there is more than one envelope in a single 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 should appear as follows:

  ```
  SOAPAction:
  ```

The Inform message contains an argument called MaxEnvelopes that indicates to the ACS the maximum number of SOAP envelopes that may be contained in a single HTTP response. The value of this parameter may be one or greater. Once the Inform message has been received, any HTTP response from the ACS may include at most this number of SOAP envelopes (requests or responses).

The Inform response also contains an argument called MaxEnvelopes that indicates to the CPE the maximum total number of SOAP envelopes that may be contained in a single HTTP post. The value of this parameter may be one or greater. Once the Inform response has been received, any HTTP post from the CPE may include at most this total number of SOAP envelopes (requests or responses).

In each direction, the order of SOAP envelopes is defined independently from the number of envelopes carried per HTTP post/response pair. Specifically, envelopes are ordered from first to last within a single HTTP post/response and then between successive post/response pairs. That is, the succession of envelopes within each HTTP post/response and then between successive posts or responses can be thought of as a single ordered sequence of envelopes.

To ensure proper association of requests and responses, the requester MAY include an ID tag in the SOAP header, which, if used, MUST be returned with the same value in the response. The encoding of this header is described in section 3.5.

Below is an example HTTP Response from an ACS containing both a Response to a prior SOAP Request, which included an ID Header, and an unrelated 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/"
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:cwmp="urn:dslforum-org:cwmp-1-0">
    <soap:Header>
        <cwmp:ID soap:mustUnderstand="1">1234</cwmp:ID>
    </soap:Header>
    <soap:Body>
        <cwmp:Response1>
            <argument>value</argument>
        </cwmp:Response1>
    </soap:Body>
</soap:Envelope>

<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:cwmp="urn:dslforum-org:cwmp-1-0">
    <soap:Body>
        <cwmp:Request2>
            <argument>value</argument>
        </cwmp:Request2>
    </soap:Body>
</soap:Envelope>
```

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.

To accommodate situations in which maintaining a continuous TCP connection is not possible (e.g., operating through an HTTP 1.0 proxy), the ACS SHOULD make use of a session cookie to maintain session state as described in [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 may 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.

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

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 MAY choose any of the following approaches in performing the transfer:

- The CPE MAY send the HTTP get/post 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 may 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, then only the latter two options are available to it.

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 ACS may issue the authentication once as part of the first HTTP transaction, and assume the authentication to hold for the duration of the TCP connection.

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 it should be a multi-part string comprising a manufacturer identifier and a serial number unique within that manufacturer. The RECOMMENDED format for this string is:

OUI-SERIAL

where OUI is a six hexadecimal-digit value using all upper-case letters and including any leading zeros. The OUI value MUST be a valid OUI as defined in [9]. SERIAL is a string that uniquely identifies the CPE from the particular manufacturer. If the manufacturer has multiple CPE products with overlapping serial number ranges, the SERIAL string MUST include additional distinguishing characters to ensure that the entire string is unique.

Example: "00D09E-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 (see section 2.2.1). 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.

3.5 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 Appendix 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:dslforum-org:cwmp-1-0”
- The data types used in Appendix A correspond directly to the data types defined in the SOAP 1.1 serialization namespace. (In general, the types used in Appendix A are restricted subsets of the corresponding SOAP types.)
- For an array argument, the given argument name corresponds to the name of the overall array element. No names are given for the individual member elements, so these should be named by their type. For example, an argument named ParameterList, which is an array of ParameterValueStruct structures, would be encoded as:

```
<ParameterList soap: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>
```

- Regarding the SOAP specification for encoding RPC methods (section 7 of [8]), for each method defined in Appendix 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” prefix to the name of the method.
- 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 SOAP `faultstring` sub-element MUST contain the string “CWMP fault”.
  - The SOAP `detail` element MUST contain a Fault structure defined in the “urn:dslforum-org:cwmp-1-0” namespace. This structure contains the following elements:
    - A `FaultCode` element that contains a single numeric fault code as defined in Appendix 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 Appendix 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.

  The following is an XML-schema segment that defines the Fault structure:

```
<xs:element Name="Fault">
  <xs:complexType>
```

```
    <xs:sequence>
      <xs:element Name="FaultCode" Type="unsignedInt"/>
      <xs:element Name="FaultString" Type="string" minOccurs="0"/>
      <xs:element Name="SetParameterValuesFault" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element Name="ParameterName" Type="string"/>
            <xs:element Name="FaultCode" Type="unsignedInt"/>
            <xs:element Name="FaultString" Type="string" minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Below is an example envelope containing a fault response:

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

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

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:cwmp="urn:dslforum-org:cwmp-1-0">
    <soap:Header>
        <cwmp:ID soap:mustUnderstand="1">1234</cwmp:ID>
    </soap:Header>
    <soap:Body>
        <soap:Fault>
            <faultcode>Client</faultcode>
            <faultstring>CWMP fault</faultstring>
            <detail>
                <cwmp:Fault>
                    <FaultCode>9003</FaultCode>
                    <FaultString>Invalid arguments</FaultString>
                    <SetParameterValuesFault>
                        <ParameterName>
                            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>
```

- For future extensibility, when processing a received envelope, both ACS and CPE MUST ignore: (a) any unknown XML elements[3] 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.

The CPE WAN Management Protocol defines a series of SOAP Header elements as specified in Table 3.

Table 3 – 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. If an ACS must update the flow-control state but has no other message to send, it may send an envelope containing only this header and an empty body. Because support for this header is required, the mustUnderstand attribute MUST be set to “1” (true) for this header.
NoMoreRequests	This header MAY be included in envelopes sent by an ACS or a CPE to explicitly indicate to the recipient whether or not it will not be sending any more requests during the remainder of the session. 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). This may be set to true in an envelope that contains the final request or in any subsequent envelope. Once set to true during a session, it SHOULD be set to true in the remaining envelopes sent, and the sender MUST NOT send additional request messages during that session. The behavior of the CPE on reception of this header is defined in section 3.7.1.4. Support in the CPE for transmission or reception of this header is OPTIONAL. The behavior of the ACS on reception of this header is defined in section 3.7.2.4. Support in the ACS for transmission or reception of this header is OPTIONAL. Because support for this header is optional, the mustUnderstand attribute MUST be either absent or set to “0” (false) 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/"
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:cwmp="urn:dslforum-org:cwmp-1-0">
    <soap:Header>
        <cwmp:ID soap:mustUnderstand="1">1234</cwmp:ID>
        <cwmp:HoldRequests soap:mustUnderstand="1">0</cwmp:HoldRequests>
        <cwmp:NoMoreRequests>1</cwmp:NoMoreRequests>
    </soap:Header>
```

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

```
    <soap:Body>
        <cwmp:Action>
            <argument>value</argument>
        </cwmp:Action>
    </soap:Body>
</soap:Envelope>
```

3.6 RPC Support Requirements

Table 4 provides a summary of all methods, and indicates the conditions under which implementation of each RPC method defined in Appendix A is REQUIRED or OPTIONAL.

Table 4 – RPC message requirements

Method name	**CPE requirement**	**Server 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[4]	Required
`Upload`	Optional	Optional
`FactoryReset`	Optional	Optional
`GetQueuedTransfers`	Optional	Optional
`ScheduleInform`	Optional	Optional
`SetVouchers`	Optional[5]	Optional
`GetOptions`	Optional	Optional
Server methods	Calling	Responding
`GetRPCMethods`	Optional	Required
`Inform`	Required	Required
`TransferComplete`	Required[6]	Required[7]
`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.

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

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

No more than one transaction session between a CPE and its associated ACS may 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.

In this initial HTTP post carrying the Inform request, only one SOAP envelope is allowed. The MaxEnvelopes argument in the Inform response indicates the maximum number of envelopes that may be carried by each subsequent HTTP post.

The CPE SHOULD initiate a session only when it has locked the Parameters accessible through this interface to ensure they cannot be changed via any other mechanism. The CPE SHOULD maintain this lock until the session is terminated.

3.7.1.2 Incoming Requests

On reception of SOAP requests from the ACS, the CPE MUST respond to each request in the order they were received, where order is defined as described in section 3.4.1. This definition of order places no constraint on whether multiple responses are sent in a single HTTP post (if the ACS can accept more than one envelope), or distributed over multiple HTTP posts.

To prevent deadlocks, the CPE MUST NOT hold off responding to an ACS request to wait for a response from the ACS to an earlier CPE request.

3.7.1.3 Outgoing Requests

When the CPE has request messages to send (after the initial Inform request), it may send these in any order with respect to responses being sent by the CPE to the ACS. That is, the CPE may insert one or more requests at any point in the sequence of envelopes it transmits to the ACS. There is no specified limit to the number of requests a CPE may send prior to receiving responses (the number of outstanding requests). A CPE MAY incorporate a locally specified limit if desired.

If the CPE receives an envelope from the ACS (either request or response) with the HoldRequests header equal to true (see section 3.5), the CPE MUST NOT send any requests in subsequent HTTP posts. The CPE may restart sending envelopes only when it subsequently receives an envelope with the HoldRequests header equal to false (or equivalently, no HoldRequests header). In determining whether it may send a request, the CPE MUST examine all envelopes received through the end of the most recent HTTP response. Because of the envelope order defined in section 3.4.1, only the last envelope in an HTTP response determines whether requests are allowed on the next HTTP post. If the CPE receives an empty HTTP response from the ACS, this may be interpreted as HoldRequests equal false.

If there are one or more outstanding requests from the ACS, or if the CPE has one or more outstanding requests and HoldRequests is false, then the CPE MUST send at least one request or response in any HTTP post sent to the ACS. An empty HTTP post MUST be sent if the ACS has no requests or responses outstanding. Table 5 lists the complete set of constraints on what a CPE MUST send while a session is in progress.

Table 5 – CPE Message Transmission Constraints

	HoldRequests	ACS requests outstanding	No ACS requests outstanding
CPE requests pending	**False**	One or more responses and/or requests	One or more requests
	True	One or more responses	Empty HTTP post
No CPE requests pending	**-**	One or more responses	Empty HTTP post

3.7.1.4 *Session Termination*

The CPE MUST terminate the transaction session when all of the following conditions are met:

1) The ACS has no further requests to send the CPE. The CPE concludes this if either one of the following is true:
    a) The most recent HTTP response from the ACS contains no envelopes.
    b) The most recent envelope received from the ACS (in the order defined in section 3.4.1) includes a NoMoreRequests header equal true (see section 3.5). Use of this header by a CPE is OPTIONAL.
2) The CPE has no further requests to send to the ACS.
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 terminate a session if it has received no HTTP response from an ACS for a locally determined time period of not less than 30 seconds.

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

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 SHOULD attempt to establish a new session, starting the session establishment procedure from the beginning. The CPE MAY place locally specified limits on the number of times it attempts to reestablish a session in this case.

3.7.2 ACS Operation

3.7.2.1 *Session Initiation*

Upon receiving the initial Inform request from the CPE, the ACS MUST respond with an Inform response. The ACS may follow this with series of requests sent to the CPE.

The MaxEnvelopes argument in the Inform request indicates the maximum number of envelopes that may be carried by each HTTP response sent by the ACS to the CPE. If the CPE can accept more than one envelope, the initial HTTP response carrying the Inform response may also carry additional requests up to the total limit imposed by MaxEnvelopes.

3.7.2.2 *Incoming Requests*

On reception of SOAP requests from the CPE, the ACS MUST respond to each request in the order they were received, where order is defined as described in section 3.4.1. This definition of order places no constraint on whether multiple responses are sent in a single HTTP response (if the CPE can accept more than one envelope), or distributed over multiple HTTP responses.

To prevent deadlocks, the ACS MUST NOT hold off responding to a CPE request to wait for a response from the CPE to an earlier ACS request.

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*

When the ACS has request messages to send, it may send these in any order with respect to responses being sent by the ACS to the CPE. That is, the ACS may insert one or more requests at any point in the sequence of envelopes it transmits to the ACS (after the Inform response). There is no specified limit to the number of requests an ACS may send prior to receiving responses (the number of outstanding requests). An ACS MAY incorporate a locally specified limit if desired.

If the ACS has one or more requests remaining to be sent and/or one or more responses outstanding from earlier requests from the CPE, the ACS MUST send at least one request or response in any HTTP response sent back to the CPE. An empty HTTP response is only allowed if the ACS has no more requests or responses outstanding.

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 may consider the session terminated when <u>all</u> of the following conditions are met:

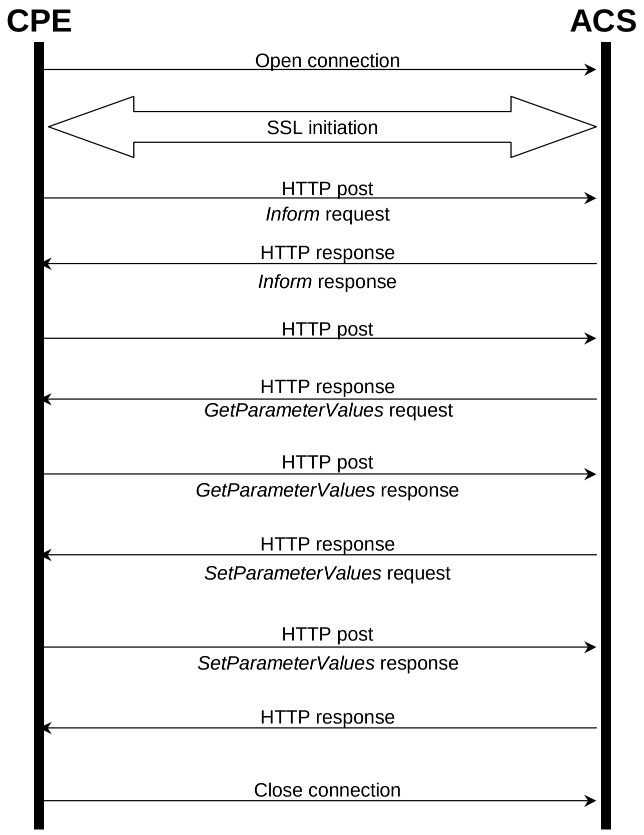
1) The CPE has no further requests to send the ACS. The ACS concludes this if either one of the following is true:
    a) The most recent HTTP post from the CPE contains no envelopes.
    b) The most recent envelope received from the CPE (in the order defined in section 3.4.1) includes a NoMoreRequests header equal true (see section 3.5). Use of this header by an ACS is OPTIONAL.
2) The ACS has no further requests to send the CPE.
3) The CPE has sent all outstanding response messages to the ACS resulting from prior requests.
4) The ACS has received all outstanding response messages from the CPE.

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).

3.7.3 Transaction Examples

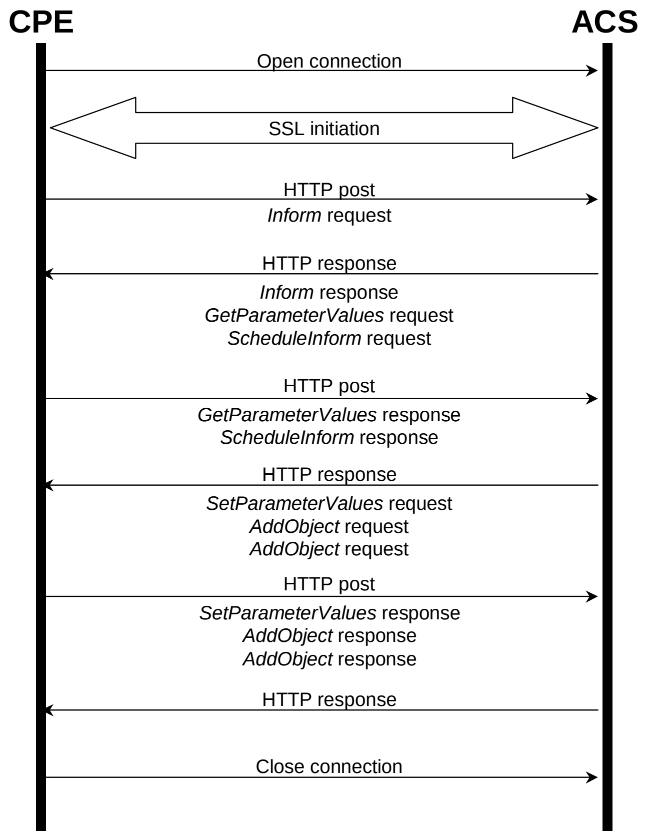
In the example shown in Figure 3, the ACS first reads a set of parameter values, and based on the result, sets some parameter values. In the example show, MaxEnvelopes from both the CPE and ACS equal one, so there is no pipelining of requests from the ACS, nor multiple responses per HTTP post from the CPE.

Figure 3 – Example with MaxEnvelopes from both the CPE and ACS equal one



The example in Figure 4 shows a scenario where MaxEnvelopes from both the CPE and ACS are equal to three, allowing the use of message pipelining in both directions. In this example, some additional requests from the ACS are shown.

Figure 4 – Example with MaxEnvelopes from both the CPE and ACS equal three



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*, DSL Forum Technical Report

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

[4] TR-044, *Auto-Configuration for Basic Internet (IP-based) Services*, DSL 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

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

[10] *The SSL Protocol, Version 3.0*, http://www.netscape.com/eng/ssl3/draft302.txt

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

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

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

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

Appendix 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 appendix 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.

It is assumed that the lower layers that transport RPC messages provide most aspects of security, including mutual authentication between the CPE and ACS, confidentiality, and data integrity.

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 6.

Table 6 – Data types

Type	Description
string	For strings listed in this specification, a maximum allowed length may 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 1 = true and 0 = false.
dateTime	The subset of the ISO 8601 date-time format defined by the SOAP dateTime type. All times are expressed in UTC (Universal Coordinated Time) unless explicitly declared otherwise. If absolute time is not available to the CPE, it SHOULD instead indicate time since boot. For example, 2 days, 3 hours, 4 minutes and 5 seconds since boot would be expressed as 0000-00-02T03:04:05.

Type	Description
base64	Base64 encoded binary. A maximum allowed length may be listed using the form base64(N), where N is the maximum length in characters after Base64 encoding.

any	An element containing any of the types listed in this table.
	Following the SOAP specification [8], elements specified as being of this type MUST include a type attribute to indicate the actual type of the element. For example:
	`<Parameter>`
	`<Name>InternetGatewayDevice.ProvisioningCode</Name>`
	`<Value xsi:type="xsd:string">code12345</Value>`
	`</Parameter>`
	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 must accommodate. A request with an array too large for the recipient to accommodate should result in the "Resources exceeded" fault code.

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 will generate an error response. Argument order must be as specified in this document.

Future versions of this specification should not redefine the RPC methods defined in this appendix. Any changes needed in a future version should 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 Servers. Either a CPE or Server may call these methods.

A.3.1.1 GetRPCMethods

This method may be used by a CPE or Server to discover the set of methods supported by the Server or CPE it is in communication with. This list may include 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. 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_00D09E_MyMethod, X_ACME_COM_MyMethod.

The calling arguments for this method are defined in Table 7. The arguments in the response are defined in Table 8.

Table 7 – GetRPCMethods arguments

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

Table 8 – GetRPCMethodsResponse arguments

Argument	Type	Description
`MethodList`	string(64)[]	Array of strings containing the names of each of the RPC methods the recipient supports. For example, a CPE implementing only the baseline methods defined in this version of the specification would return the following list when requested by a Server: "GetRPCMethods" "SetParameterValues" "GetParameterValues" "GetParameterNames" “SetParameterAttributes” “GetParameterAttributes” “AddObject” “DeleteObject” “Reboot” “Download”

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 a Server can call these methods.

A.3.2.1 SetParameterValues

This method may be used by a Server to modify the value of one or more CPE Parameters. The calling arguments for this method are defined in Table 9. The arguments in the response are defined in Table 10.

Table 9 – SetParameterValues arguments

Argument	Type	Description
`ParameterList`	ParameterValueStruct[]	Array of name-value pairs as specified in Table 11. For each name-value pair, the CPE is instructed to set the Parameter specified by the name to the corresponding value.
`ParameterKey`	string(32)	The value to set the ParameterKey parameter. This MAY be used by the server to identify Parameter updates, or left empty.

Table 10 – SetParameterValuesResponse arguments

Argument	Type	Description
`Status`	int[0:1]	A successful response to this method returns an integer enumeration defined as follows: 0 = Parameter changes have been validated and applied. 1 = Parameter changes have been validated and committed, but 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 each of the specified Parameters immediately and atomically. 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. A successful response to this RPC SHOULD occur only after the new Parameter values have been successfully applied. If the CPE requires a reboot before applying the Parameter values, the CPE MUST reply before such a reboot, and thus before the Parameter values have been applied. 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 immediately following the reboot.

The `ParameterValueStruct` structure is defined in Table 11.

Table 11 – ParameterValueStruct definition

Name	Type	Description
`Name`	string(256)	This is the name of a Parameter.
`Value`	any	This is the value the Parameter is to be set.

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

A.3.2.2 GetParameterValues

This method may be used by a Server to obtain 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 – 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 13 – GetParameterValuesResponse arguments

Argument	Type	Description
`ParameterList`	ParameterValueStruct[]	Array of name-value pairs, as specified in Table 11, containing the name and value for each requested Parameter.

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.3.2.3 GetParameterNames

This method may be used by a Server to discover the Parameters accessible on a particular CPE. The calling arguments for this method are defined in Table 14. The arguments in the response are defined in Table 15.

Table 14 – 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.
`NextLevel`	boolean	If false, the response lists the full path name of all Parameters whose name begins with the string given by the `ParameterPath` argument. If true, the response lists only the partial path one level below the specified ParameterPath. For example, if ParameterPath is "InternetGatewayDevice.LAN-Device.", the response may list "InternetGatewayDevice.LANDevice.1." and "InternetGatewayDevice.LANDevice.2." without listing all of the Parameters below this level.

Table 15 – GetParameterNamesResponse arguments

Argument	Type	Description
`ParameterList`	ParameterInfoStruct[]	Array of structures, each containing the name and other information for a Parameter, as defined in Table 16. When NextLevel is false, this method returns Parameter information for all accessible Parameters whose name begins with the string given by the `ParameterPath` argument. If the `ParameterPath` argument is an empty string, names of all Parameters accessible on the particular CPE are returned. When NextLevel is true, this list contains all partial paths one level below the path specified in ParameterPath.

Table 16 – ParameterInfoStruct definition

Name	Type	Description
`Name`	string(256)	This is the name of a Parameter or partial path name.
`Writable`	boolean	Whether or not the Parameter value can be overwritten using the SetParameterValues method. If Name is a partial path due to NextLevel being true, this indicates whether AddObject and DeleteObject can be used at this level to remove this instance or add other instances.

The following fault codes are defined for this method: 9001, 9002, 9003, 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.3.2.4 SetParameterAttributes

This method may be used by a Server to modify attributes associated with one or more CPE Parameter. The calling arguments for this method are defined in Table 17. The arguments in the response are defined in Table 18.

Table 17 – 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 19.

Table 18 – SetParameterAttributesResponse arguments

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

Table 19 – 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. 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 should include changed values of the specified parameter(s) in the Inform message, and whether the CPE must 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. Whenever a parameter change is sent in the Inform message due to a non-zero Notification setting, the Event code "4 VALUE CHANGE" MUST be included in the list of Events. The CPE may return a "notification request rejected" error if an attempt is made to set notification on a parameter deemed inappropriate (e.g., a continuously varying statistic).
`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, 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 a device controlled by the subscriber on the LAN, such as via the LAN-Side DSL CPE Configuration protocol or via UPnP. By default, prior to any changes to the access list by an ACS, access SHOULD be granted to all entities specified above.

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).

A.3.2.5 GetParameterAttributes

This method may be used by a Server to read the 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.

Table 20 – 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 21 – 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 ParameterAccessStruct as defined in Table 22.

Table 22 – ParameterAttributesStruct definition

Name	Type	Description
`Name`	string(256)	This is the name of a Parameter to which the attributes are given.

Name	Type	Description
`Notification`	int[0:2]	Indicates whether the CPE should include changed values of the specified parameter(s) in the Inform message, and whether the CPE must 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.
`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, 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 a device controlled by the subscriber on the LAN, such as via the LAN-Side DSL CPE Configuration protocol or via UPnP.

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 Server 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 refereced 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 may 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 are set to their default values and the associated attributes are set to the following:

- Notification is set to zero (notification off)
- AccessList includes all defined entities

For an Internet Gateway Device, the specific set of objects for which this method may be used is listed in Appendix B. The calling arguments for this method are defined in Table 23. The arguments in the response are defined in Table 24.

Table 23 – 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 value of this argument is left to the discretion of the server, and may be left empty.

Table 24 – 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 may 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.
`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.

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 may discard the state previously associated with any Parameter or sub-object 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.

For an Internet Gateway Device, the specific set of objects for which this method may be used is listed in Appendix B. The calling arguments for this method are defined in Table 25. The arguments in the response are defined in Table 26.

Table 25 – 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 value of this argument is left to the discretion of the server, and may be left empty.

Table 26 – 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.

A.3.2.8 Download

This method may be used by the Server to cause the CPE to download a specified file from the designated location. The calling arguments for this method are defined in Table 27. The arguments in the response are defined in Table 28.

Table 27 – 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 TransferComplete and GetQueuedTransfers.
`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.
`URL`	string(256)	URL specifying the source file location. HTTP transport MUST be supported. Other optional transports, as specified in section 2.3.2, MAY be supported.
`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 CPE may use this value to determine if it has sufficient space for the file, or if it must free up additional space to make room for the specified file.
`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 SHOULD 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).
`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 SHOULD NOT occur in the same transaction session in which the request was issued.

Argument	Type	Description
`SuccessURL`	string(256)	When applicable, this argument contains the URL the CPE should redirect the user's browser to if the download completes successfully. This URL may include CGI arguments as needed by the Server (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 the CPE should redirect the user's browser to if the download does not complete successfully. This URL may include CGI arguments as needed by the Server (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 28 – 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 completed (for example, if the CPE needs to reboot itself before it can perform the file download). 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.
`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.

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

A.3.2.9 Reboot

This method causes the CPE to reboot. The CPE MUST send the method response prior to rebooting. It should be used with extreme caution. The calling arguments for this method are defined in Table 29. The arguments in the response are defined in Table 30.

> *Note – this method is primarily intended for diagnostic purposes. This method is not intended for use by an ACS to initiate a reboot after setting CPE parameters or initiating a download. If a CPE requires a reboot in those cases, it is responsible for initiating that reboot on its own after the termination of the session. Because some CPE may not require a reboot under these circumstances, an ACS should not call this method in these cases, which would result in an unnecessary reboot.*

Table 29 – 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.

Table 30 – 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 Server Methods

The methods listed in this section are defined to be supported on a Server. Only a CPE can call these methods.

A.3.3.1 Inform

A CPE MUST call the `Inform` method to initiated a transaction sequence whenever a connection to an ACS is established. The calling arguments for this method are defined in Table 31. The arguments in the response are defined in Table 32.

Table 31 – Inform arguments

Argument	Type	Value
`DeviceId`	DeviceIdStruct	A structure that uniquely identifies the CPE, defined in Table 34.
`Event`	EventStruct[16]	An array of structures, as defined in Table 35, indicating one or more events that caused the transaction session to be established. If one or more causes exist, the CPE MUST list all such causes.
`MaxEnvelopes`	unsignedInt	Maximum number of SOAP envelopes in a single HTTP response that the CPE will accept from the ACS. A value of zero indicates that there is no specific limit on the number of envelopes.
`CurrentTime`	dateTime	The current date and time known to the CPE in UTC.
`RetryCount`	unsignedInt	Number of prior times an attempt was made to call the Inform method before it was successfully achieved. Specifically, this value is incremented for each unsuccessful attempt either to establish a connection to the ACS for the purpose of sending an Inform message, and for each unsuccessful use of the Inform message in which an error response or no response was received. This value is reset to zero after an Inform message was sent with a successful response.
`ParameterList`	ParameterValueStruct[]	Array of name-value pairs as specified in Table 11. The Inform request contains a list of informational parameters, specific to a given type of CPE. The parameters that MUST be included in the case of an Internet Gateway Device are listed in Table 33. A CPE MAY send additional parameters as well.

Table 32 – InformResponse arguments

Argument	Type	Description
`MaxEnvelopes`	unsignedInt	Maximum number of SOAP envelopes in a single HTTP post that the ACS will accept from the CPE. A value of zero indicates that there is no specific limit on the number of envelopes.

Table 33 lists the parameters required to be present in an Inform from an Internet Gateway Device. The CPE MAY include additional parameters in the Inform as well.

When the Inform call results from a change to one or more parameter values (due to a cause other than being set by the ACS itself) that the ACS has marked for notification (either active or passive) via SetParameterAttributes, all of the changed parameters must also be included in the ParameterList. If a parameter has changed more than once since the last such notification, only the most recent parameter value is given.

For the items marked in the Inform on Change column of Table 33, any change in the value of the parameter MUST result in the CPE initiating a connection to the ACS to issue the Inform method call, regardless of whether the ACS has marked these parameters for notification.

An Inform call resulting from a change to a parameter marked in Table 33 as Inform on Change, or a parameter the ACS has marked for notification (either active or passive), MUST include the Event "4 VALUE CHANGE" in the Event list.

Table 33 – Required Inform parameters for an Internet Gateway Device

Parameter	Inform on Change
InternetGatewayDevice.DeviceInfo.SpecVersion	
InternetGatewayDevice.DeviceInfo.HardwareVersion	
InternetGatewayDevice.DeviceInfo.SoftwareVersion	X
InternetGatewayDevice.DeviceInfo.ProvisioningCode	X
InternetGatewayDevice.ManagementServer.ConnectionRequestURL	X
InternetGatewayDevice.ManagementServer.ParameterKey	
InternetGatewayDevice.WANDevice.{i}.WANConnectionDevice.{j}.WAN{***}Connection.{k}.-ExternalIPAddress[8]	X[9]

Table 34 – 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].
`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 35 – EventStruct definition

Name	Type	Description
`EventCode`	string(64)	Each value consists of an identifying character followed by a text description of the cause. Several specific cause values are listed below, as well as a syntax for specifying vendor-specific causes.

Value	Cause
"0 BOOTSTRAP"	Indicates that the session was established due to first-time CPE installation or a change to the ACS URL.

[8] Where {i}, {j}, and {k} refer to the default WAN connection, and {***} is either “IP” or “PPP” depending on the type of connection.

[9] The CPE must initiate an Inform whenever either the value of this parameter changes or the default WAN connection changes to a different connection.

Name	Type	Description	
		"1 BOOT"	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.
		"2 PERIODIC"	Indicates that the session was established on a periodic Inform interval.
		"3 SCHEDULED"	Indicates that the session was established due to a ScheduleInform method call.
		"4 VALUE CHANGE"	Indicates that the session was established due to a change to one of the Parameter values included in the Inform method call. For example, the allocation of a new IP address to the CPE.
		"5 KICKED"	Indicates that the session was established for the purpose of web identity management (see Appendix D) and that a Kicked method (see section A.4.2.1) will be issued during this session.
		“6 CONNECTION REQUEST”	Indicates that the session was established due to a Connection Request notification from the Server as described in section 3.2.
		“7 TRANSFER COMPLETE”	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.
		"8 DIAGNOSTICS COMPLETE"	Used when reestablishing a connection to the ACS after completing a diagnostic test initiated by the ACS. For example, the DSL loop diagnostics (see Appendix B).
		"M "<method name>	If this results from another method, the value is a “M” followed by a space and the method name. For example: “M Reboot”
		“X “<OUI> <event>	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]. The value and interpretation of <event> is vendor-specific. For example: “X 00D09E MyEvent”

Name	Type	Description
`CommandKey`	string(32)	If the Inform structure is generated in response to a cause in which a CommandKey has been specified, this element MUST contain the value of that CommandKey. In all other cases, this element is an empty string. 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 • Reboot method • Download method • Upload method

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

A.3.3.2 TransferComplete

This method informs the server 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. When used, this method should be called only after the transfer has completed (or failed). The criteria used by a CPE to determine when a transfer is considered complete are specific to the implementation of the CPE. The calling arguments for this method are defined in Table 36. The arguments in the response are defined in Table 37.

Table 36 – 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 38. 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.
`CompleteTime`	dateTime	The date and time transfer completed in UTC.

Table 37 – TransferCompleteResponse arguments

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

Table 38 – 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 a Server can call these methods.

A.4.1.1 GetQueuedTransfers

This method may be used by a Server to determine the status of previously requested downloads or uploads. The calling arguments for this method are defined in Table 39. The arguments in the response are defined in Table 40.

Table 39 – GetQueuedTransfers arguments

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

Table 40 – GetQueuedTransfersResponse arguments

Argument	Type	Description
`TransferList`	QueuedTransferStruct[16]	Array of structures as defined in Table 41, each describing the state of one transfer that the CPE has been instructed to perform, but has not yet been fully completed.

Table 41 – 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 a Server 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 42. The arguments in the response are defined in Table 43.

Table 42 – 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.

Table 43 – 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 a Server to set one or more option Vouchers in the CPE. The calling arguments for this method are defined in Table 44. The arguments in the response are defined in Table 45.

Table 44 – 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 Appendix C.

Table 45 – 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 a Server 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 46. The arguments in the response are defined in Table 47.

Table 46 – 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 47 – GetOptionsResponse arguments

Argument	Type	Description
`OptionList`	OptionStruct[]	Array of OptionStructs as defined in Table 48, 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 48 – OptionStruct definition

Name	Type	Description
`OptionName`	string(64)	Identifying name of the particular Option.
`VoucherSN`	unsignedInt	Identifying number of the particular Option.

Name	Type	Description
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.
ExpirationDate	dateTime	The specified date the Option is to expire in UTC, if any.
IsTransferable	boolean	Indicates whether or not the Option has been designated transferable or non-transferable (see Appendix 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 Server to cause the CPE to upload a specified file to the designated location. The calling arguments for this method are defined in Table 49. The arguments in the response are defined in Table 50.

Table 49 – 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 TransferComplete and GetQueuedTransfers.
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.
URL	string(256)	URL specifying the destination file location. HTTP transport MUST be supported. Other optional transports, as specified in section 2.3.2, MAY be supported.
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.

Argument	Type	Description
`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 SHOULD NOT occur in the same transaction session in which the request was issues.

Table 50 – 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 must 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.
`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.

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

A.4.1.6 FactoryReset

This method resets the CPE to its factory default state. This method should be used with extreme caution. The calling arguments for this method are defined in Table 51. The arguments in the response are defined in Table 52.

Table 51 – FactoryReset arguments

Argument	Type	Description
-	void	This method has no arguments.

Table 52 – 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 Server Methods

The methods listed in this section may optionally be supported on a Server. 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 Appendix D. The calling arguments for this method are defined in Table 53. The arguments in the response are defined in Table 54.

Table 53 – Kicked arguments

Argument	Type	Value
`Command`	string(32)	Generic argument that may be used by the Server 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 Server for identification or other purposes.
`Next`	string(1024)	The URL the Server should return in the method response under normal conditions.

Table 54 – 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 as needed by the Server (for example, to maintain session state). If the Server 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 Server. On reception of this request, the Server MAY call the `Download` method to initiate the download. The calling arguments for this method are defined in Table 55. The arguments in the response are defined in Table 56.

Table 55 – RequestDownload arguments

Argument	Type	Value
`FileType`	string(64)	This is the FileType being requested (see Table 27 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 57. The use of the additional arguments depend on the FileType specified. The following arguments are defined for each of the currently defined file types. **FileType** / **FileTypeArg Names**: 1 Firmware Upgrade / (none) 2 Web Content / "Version" 3 Vendor Configuration File / (none) If the Server receives arguments that it does not understand, it MUST ignore the unknown arguments, but process the request using the arguments that it does understand.

Table 56 – RequestDownloadResponse arguments

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

Table 57 – 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 58 lists the fault codes that can be returned by a CPE. Note that the fault code values are shown in decimal representation.

Table 58 – Fault codes

Fault code	Description
`9000`	Method not supported
`9001`	Request denied (no reason specified)
`9002`	Internal error
`9003`	Invalid arguments
`9004`	Resources exceeded (when used in association with SetParameterValues, this MUST not be used to indicate parameters in error)
`9005`	Invalid parameter name (associated with Set/GetParameterValues, GetParameterNames, Set/GetParameterAttributes)
`9006`	Invalid parameter type (associated with SetParameterValues)
`9007`	Invalid parameter value (associated with SetParameterValues)
`9008`	Attempt to set a non-writable parameter (associated with SetParameterValues)
`9009`	Notification request rejected (associated with SetParameterAttributes method).
`9010`	Download failure (associated with Download or TransferComplete methods).
`9011`	Upload failure (associated with Upload or TransferComplete methods).
`9012`	File transfer server authentication failure (associated with Upload, Download, or TransferComplete methods).
`9013`	Unsupported protocol for file transfer (associated with Upload and Download methods).
`9800 – 9899`	Vendor defined fault codes

A.5.2 Server Fault Codes

Table 59 lists the fault codes that can be returned by a Server. Note that the fault code values are shown in decimal representation.

Table 59 – Fault codes

Fault code	Description
`8000`	Method not supported
`8001`	Request denied (no reason specified)
`8002`	Internal error
`8003`	Invalid arguments
`8004`	Resources exceeded
`8005`	Retry request

Fault code	Description
8800 – 8899	Vendor defined fault codes

A.5.3 Server Method Retry Behavior

The CPE SHOULD retry each Server method if it is unsuccessful. When retrying an unsuccessful method call, the CPE SHOULD use an exponential back-off algorithm in determining the retry interval.

The CPE MUST immediately retry a method call if it receives a “Retry request” (fault code 8005) in the response.

Appendix B. CPE Parameters

B.1 Introduction

This appendix defines all of the CPE Parameters for an Internet Gateway Device that are accessible via the RPC Parameter-related methods as defined in Appendix A. This list includes Parameters that are REQUIRED in the CPE as well as those that are OPTIONAL.

B.2 CPE Parameters

Table 61 lists all defined Parameters for an Internet Gateway Device that are accessible via the RPC methods SetParameterValues and GetParameterValues, and GetParameterNames.

The table defines the name, the data type, whether or not the Parameter is write-accessible, whether or not the Parameter is REQUIRED (R), OPTIONAL (O), or CONDITIONAL (C) for reading or writing, and a description. Unless otherwise specificed, CONDITIONAL implies that the Parameter is required if the object containing it is implemented at all.

Parameter names use a hierarchical form similar to a directory tree. The name of a particular Parameter is represented by the concatenation of each successive node in the hierarchy separated with a "." (dot), starting at the trunk of the hierarchy and leading to the leaves. When specifying a partial path, indicating an intermediate node in the hierarchy, the trailing "." (dot) is always used as the last character.

In some cases, where multiple instances of an object can occur, the placeholder node name "{i}" is shown. In actual use, this placeholder is to be replaced by an instance number, which MUST be a positive integer (≥ 1). For instances of an object that are created using the AddObject method, this instance number corresponds to the InstanceNumber argument returned by this method upon creation. Because in some cases object instances may also be deleted, instance numbers will in general not be contiguous.

While this document only specifies the parameter list for an Internet Gateway Device, several of the objects are expected to be usable for other types of devices. In such cases, the path name of each parameter should match that in Table 61 with the exception that the actual device type would replace "InternetGateway-Device" in the name. The following objects are expected to be usable in most other device types:

- DeviceInfo
- DeviceConfig
- ManagementServer
- Time

B.2.1 Data Types

The parameters defined in this specification make use of a limited subset of the default SOAP data types [8]. The complete set of parameter data types along with the notation used to represent these types is listed in Table 60.

Table 60 – Data types

Type	Description
object	A container for parameters and/or other objects. The full path name of a parameter is given by the parameter name appended to the full path name of the object it is contained within.
string	For strings listed in this specification, a maximum allowed length may 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.
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 1 = true and 0 = false.
dateTime	The subset of the ISO 8601 date-time format defined by the SOAP dateTime type. All times are expressed in UTC (Universal Coordinated Time) unless explicitly declared otherwise. If absolute time is not available to the CPE, it SHOULD instead indicate time since boot. For example, 2 days, 3 hours, 4 minutes and 5 seconds since boot would be expressed as 0000-00-02T03:04:05.
base64	Base64 encoded binary. A maximum allowed length may be listed using the form base64(N), where N is the maximum length in characters after Base64 encoding.

All IP addresses and subnet masks are represented as strings in IPv4 dotted-decimal notation. Note that there is no IPv6 support at this time in the parameter list described in this appendix.

All MAC addresses are represented as strings of 12 hexadecimal digits (digits 0-9, letters A-F or a-f) displayed as six pairs of digits separated by colons.

B.2.2 Vendor-Specific Parameters

A vendor may extend the standardized parameter list with vendor-specific parameters and objects. Vendor-specific parameters and objects may be defined either in a separate naming hierarchy or within the standardized naming hierarchy.

The name of a vendor-specific parameter or object MUST have the form:

X_<VENDOR>_VendorSpecificName

In this definition <VENDOR> is a unique vendor identifier, which may be either an OUI or a domain name. An OUI is an organizationally unique identifier as defined in [9], which MUST formatted as a six-hexadecimal-digit string using all upper-case letters and including any leading zeros. A domain name MUST be upper case with each dot (".") replaced with a hyphen or underscore.

The VendorSpecificName MUST be a valid string as defined in B.2.1, and MUST NOT contain a "." (period) or a space character.

Note – the use of the string "X_" to indicate a vendor-specific parameter implies that no standardized parameter can begin with "X_".

Below are some example vendor-specific parameter and object names:

InternetGatewayDevice.UserInterface.X_00D09E_AdBanner

InternetGatewayDevice.LANDevice.1.X_00D09E_LANInfraredInterfaceConfig.2.Status

X_GAMECO-COM_GameDevice.Info.Type

When appropriate, a vendor may also extend the set of values of an enumeration. If this is done, the vendor-specified values must be in the form “X_<VENDOR>_VendorSpecificValue”. The total length of such a string MUST NOT exceed 31 characters.

B.2.3 Parameter List

Table 61 – CPE Parameter List for an Internet Gateway Device

Name[10]	Type	Write[11]	Read	Description
InternetGatewayDevice.	object	-[12]	R[13]	The top-level object for an Internet Gateway Device.
LANDeviceNumberOfEntries	unsignedInt	-	R	Number of instances of LANDevice.
WANDeviceNumberOfEntries	unsignedInt	-	R	Number of instances of WANDevice.
InternetGatewayDevice.DeviceInfo.	object	-	R	This object contains general device information.
Manufacturer	string(64)	-	R	The manufacturer of the CPE (human readable string).
ManufacturerOUI	string(6)	-	R	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].
ModelName	string(64)	-	R	Model name of the CPE (human readable string).
Description	string(256)	-	R	A full description of the CPE device (human readable string).
ProductClass	string(64)	-	O	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)	-	R	Serial number of the CPE.
HardwareVersion	string(64)	-	R	A string identifying the particular CPE model and version.
SoftwareVersion	string(64)	-	R	A string identifying the software version currently installed in the CPE. To allow version comparisons, this element SHOULD be in the form of dot-delimited integers, where each successive integer represents a more minor category of variation. For example, 3.0.21where the components mean: Major.Minor.Build.

[10] The full name of a Parameter is the concatenation of the object name shown in the yellow header with the individual Parameter name.

[11] “R” = Required, “O” = Optional, “C” = Conditional, “-” = Not present

[12] Write access for an object indicates whether the AddObject and DeleteObject actions are not allowed (“-”), optionally allowed (“O”), required (“R”), or conditionally required if the object is supported at all (“C”).

[13] Read access for an object indicates whether or not the object is optional (“O”), required (“R”), or conditionally required (“C”) if the CPE supports the related functionality or if the object containing it is present.

Name[10]	Type	Write[11]	Read	Description
ModemFirmwareVersion	string(64)	-	O	A string identifying the version of the modem firmware currently installed in the CPE. This is applicable only when the modem firmware is separable from the overall CPE software.
EnabledOptions	string-(1024)	-	O	Comma-separated list of the OptionName of each Option that is currently enabled in the CPE. The OptionName of each is identical to the OptionName element of the OptionStruct described in Table 48. Only those options are listed whose State indicates the option is enabled.
AdditionalHardwareVersion	string(64)	-	O	A comma separated list of any additional versions. Represents any additional hardware version information the vendor may wish to supply.
AdditionalSoftwareVersion	string(64)	-	O	A comma separated list of any additional versions. Represents any additional software version information the vendor may wish to supply.
SpecVersion	string(16)	-	R	Represents the version of the specification implemented by the device. Currently 1.0 is the only available version.
ProvisioningCode	string(64)	R	R	Identifier of the primary service provider and other provisioning information, which MAY be used by the Server to determine service provider-specific customization and provisioning parameters. If non-empty, this argument SHOULD be in the form of a hierarchical descriptor with one or more nodes specified. Each node in the hierarchy is represented as a 4-character sub-string, containing only numerals or upper-case letters. If there is more than one node indicated, each node is separated by a "." (dot). Examples: “TLCO” or “TLCO.GRP2”.
UpTime	unsignedInt	-	R	Time in seconds since the CPE was last restarted.
FirstUseDate	dateTime	-	O	Date and time in UTC that the CPE first successfully established a network connection.
DeviceLog	string(32K)	-	R	Vendor-specific log(s).
VendorConfigFileNumberOfEntries	unsignedInt	-	O	Number of instances of VendorConfigFile.
InternetGatewayDevice.DeviceInfo.Vendor-ConfigFile.{i}.	object	-	O	Every instance of this object is a Vendor Configuration File, and contains parameters associated with the Vendor Configuration File.
Name	string(64)	-	C	Name of the vendor configuration file.
Version	string(16)	-	C	A string identifying the configuration file version currently used in the CPE.
Date	dateTime	-	C	Date and time when the content of the current version of this vendor configuration file was first applied by the CPE.
Description	string(256)	-	O	A description of the vendor configuration file (human-readable string).
InternetGatewayDevice.DeviceConfig.	object	-	O	This object contains general configuration parameters.
PersistentData	string(256)	O	O	Arbitrary user data that must persist across CPE reboots.
ConfigFile	string(32K)	O	O	A dump of the currently running configuration on the CPE. This parameter enables the ability to backup and restore the last known good state of the CPE. It returns a vendor-specific document that defines the state of the CPE. The document must be capable of restoring the CPE’s state when written back to the CPE using SetParameterValues.

Name[10]	Type	Write[11]	Read	Description
InternetGatewayDevice.ManagementServer.	object	-	R	This object contains parameters relating to the CPE's association with an ACS.
URL	string(256)	R	R	URL for the CPE to connect to the ACS using the CPE WAN Management Protocol. This parameter MUST be in the form of a valid HTTP or HTTPS URL [5]. An HTTPS URL indicates that the ACS supports SSL. The "host" portion of this URL is used by the CPE for validating the certificate from the ACS when using certificate-based authentication.
Username	string(256)	R	-	Username used to authenticate the CPE when making a connection to the ACS using the CPE WAN Management Protocol. This username is used only for HTTP-based authentication of the CPE.
Password	string(256)	R	-	Password used to authenticate the CPE when making a connection to the ACS using the CPE WAN Management Protocol. This password is used only for HTTP-based authentication of the CPE. When read, this parameter returns an empty string, regardless of the actual value.
PeriodicInformEnable	boolean	R	R	Whether or not the CPE must periodically send CPE information to Server using the Inform method call.
PeriodicInformInterval	unsignedInt [1:]	R	R	The duration in seconds of the interval for which the CPE MUST attempt to connect with the ACS and call the Inform method if PeriodicInform-Enable is true.
PeriodicInformTime	dateTime	R	R	An absolute time reference in UTC to determine when the CPE should initiate the Inform method calls. Each Inform call must occur at this reference time plus or minus an integer multiple of the PeriodicInformInterval. A zero dateTime value (0000-00-00T00:00:00) indicates that no particular time reference is specified. That is, the CPE may locally choose the time reference, required only to adhere to the specified PeriodicInformInterval.
ParameterKey	string(32)	-	R	The value of the ParameterKey argument from the most recent SetParameterValues, AddObject, or DeleteObject method call from the Server. If there have been no such calls, this value is empty.
ConnectionRequestURL	string(256)	-	R	HTTP URL for an ACS to make a Connection Request notification to the CPE. In the form: http://host:port/path The "host" portion of the URL MAY be the IP address for the management interface of the CPE in lieu of a host name.
ConnectionRequestUsername	string(256)	R	R	Username used to authenticate an ACS making a Connection Request to the CPE.
ConnectionRequestPassword	string(256)	R	-	Password used to authenticate an ACS making a Connection Request to the CPE. When read, this parameter returns an empty string, regardless of the actual value.

Name[10]	Type	Write[11]	Read	Description
UpgradesManaged	boolean	R	R	Indicates whether or not the ACS will manage upgrades for the CPE. If true (1), the CPE SHOULD not use other means other than the ACS to seek out available upgrades. If false (0), the CPE MAY use other means for this purpose.
KickURL	string(256)	-	O	Present only for a CPE that supports the Kicked RPC method. LAN-accessible URL from which the CPE can be “kicked” to initiate the Kicked RPC method call. MUST be an absolute URL including a host name or IP address as would be used on the LAN side of the CPE.
DownloadProgressURL	string(256)	-	O	Present only for a CPE that provides a LAN-side web page to show progress during a file download. LAN-accessible URL to which a web-server associated with the ACS may redirect a user’s browser on initiation of a file download to observer the status of the download.
InternetGatewayDevice.Time.	object	-	O	This object contains parameters relating an NTP or SNTP time client in the CPE. Support for this object is Optional.
NTPServer1	string(64)	C	C	First NTP timeserver. Either a host name or IP address.
NTPServer2	string(64)	C	C	Second NTP timeserver. Either a host name or IP address.
NTPServer3	string(64)	O	O	Third NTP timeserver. Either a host name or IP address.
NTPServer4	string(64)	O	O	Fourth NTP timeserver. Either a host name or IP address.
NTPServer5	string(64)	O	O	Fifth NTP timeserver. Either a host name or IP address.
CurrentLocalTime	dateTime	-	C	The current date and time in the CPE’s local time zone.
LocalTimeZone	string(6)	C	C	The local time offset from UTC in the form: +hh:mm -hh:mm
LocalTimeZoneName	string(64)	C	C	Name of the local time zone (human readable string).
DaylightSavingsUsed	boolean	C	C	Whether or not daylight savings time is in use in the CPE’s local time zone.
DaylightSavingsStart	dateTime	C	C	Date and time daylight savings time begins if used in local standard time. If daylight savings time is not used, this value is ignored.
DaylightSavingsEnd	dateTime	C	C	Date and time daylight savings time ends if used in local daylight time. If daylight savings time is not used, this value is ignored.
InternetGatewayDevice.UserInterface.	object	-	O	This object contains parameters relating to the user interface of the CPE. Support for this object is Optional.
PasswordRequired	boolean	O	O	Present only if the CPE provides a password-protected LAN-side user interface. Indicates whether or not the local user interface must require a password to be chosen by the user. If false, the choice of whether or not a password is used is left to the user.

Name[10]	**Type**	**Write**[11]	**Read**	**Description**
PasswordUserSelectable	boolean	O	O	Present only if the CPE provides a password-protected LAN-side user interface and supports LAN-side Auto-Configuration. Indicates whether or not a password to protect the local user interface of the CPE may be selected by the user directly, or must be equal to the password used by the LAN-side Auto-Configuration protocol.
UpgradeAvailable	boolean	O	O	Indicates that a CPE upgrade is available, allowing the CPE to display this information to the user.
WarrantyDate	dateTime	O	O	Indicates the date and time in UTC that the warranty associated with the CPE is to expire.
ISPName	string(64)	O	O	The name of the customer's ISP.
ISPHelpDesk	string(32)	O	O	The help desk phone number of the ISP.
ISPHomePage	string(256)	O	O	The URL of the ISP's home page.
ISPHelpPage	string(256)	O	O	The URL of the ISP's on-line support page.
ISPLogo	base64 (5460)	O	O	Base64 encoded GIF or JPEG image. The binary image is constrained to 4095 bytes or less.
ISPLogoSize	unsignedInt [0:4095]	O	O	Un-encoded binary image size in bytes. If ISPLogoSize input value is 0 then the ISPLogo is cleared. ISPLogoSize can also be used as a check to verify correct transfer and conversion of Base64 string to image size.
ISPMailServer	string(256)	O	O	The URL of the ISP's mail server.
ISPNewsServer	string(256)	O	O	The URL of the ISP's news server.
TextColor	string(6)	O	O	The color of text on the GUI screens in RGB hexidecimal notation (e.g., FF0088).
BackgroundColor	string(6)	O	O	The color of the GUI screen backgrounds in RGB hexidecimal notation (e.g., FF0088).
ButtonColor	string(6)	O	O	The color of buttons on the GUI screens in RGB hexidecimal notation (e.g., FF0088).
ButtonTextColor	string(6)	O	O	The color of text on buttons on the GUI screens in RGB hexidecimal notation (e.g., FF0088).
AutoUpdateServer	string(256)	O	O	The server the CPE can check to see if an update is available for direct download to it. This MUST NOT be used by the CPE if the InternetGateway-Device.ManagementServer.UpgradesManaged parameter is true (1).
UserUpdateServer	string(256)	O	O	The server where a user can check via a web browser if an update is available for download to a PC. This MUST NOT be used by the CPE if the InternetGatewayDevice.ManagementServer.-UpgradesManaged parameter is true (1).
ExampleLogin	string(40)	O	O	An example of a correct login, according to ISP-specific rules.
ExamplePassword	string(30)	O	O	An example of a correct password, according to ISP-specific rules.
InternetGatewayDevice.Layer3Forwarding.	object	-	R	This object allows the handling of the routing and forwarding configuration of the device.
DefaultConnectionService	string(256)	R	R	Specifies the default WAN interface. The content is the full hierarchical parameter name of the default layer-3 connection object. Example: "InternetGatewayDevice.WANDevice.1.WAN-ConnectionDevice.2.WANPPPConnection.1".
ForwardNumberOfEntries	unsignedInt	-	R	Number of forwarding instances.

Name[10]	Type	Write[11]	Read	Description
InternetGatewayDevice.Layer3Forwarding.-Forwarding.{i}.	object	R	R	Layer-3 forwarding table.
Enable	boolean	R	R	Enables or disables the forwarding entry. On creation, an entry is disabled by default.
Status	string	-	R	Indicates the status of the forwarding entry. Enumeration of: “Disabled” “Enabled” “Error”
Type	string	R	R	Indicates the type of route. Enumeration of: “Default” “Network” “Host”
DestIPAddress	string	R	R	Destination address.
DestSubnetMask	string	R	R	Destination subnet mask.
SourceIPAddress	string	R	R	Source address.
SourceSubnetMask	string	R	R	Source subnet mask.
GatewayIPAddress	string	R	R	IP address of the gateway.
Interface	string	R	R	Specifies the WAN interface associated with this entry. The content is the full hierarchical parameter name of the layer-3 connection object. Example: “InternetGatewayDevice.WANDevice.1.-WANConnectionDevice.2.WANPPPConnection.-1”.
ForwardingMetric	int[-1:]	R	R	Forwarding metric. A value of -1 indicates this metric is not used.
MTU	unsignedInt [1:1540]	O	O	The maximum allowed size of an Ethernet frame for this route.
InternetGatewayDevice.LANConfigSecurity.	object	-	R	This object contains generic device configuration information.
ConfigPassword	string(64)	R	-	A password to allow LAN access to protected auto-configuration services. When read, this parameter returns an empty string, regardless of the actual value.

Name[10]	Type	Write[11]	Read	Description
InternetGatewayDevice.IPPingDiagnostics	object	-	O	This object is provides access to an IP-layer ping test.
DiagnosticsState	string	C	C	Indicates availability of diagnostic data. One of: "None" "Requested" "Complete" "Error_CannotResolveHostName" Value may be set to Requested to initiate the diagnostic test. When writing, the only allowed value is Requested. To ensure the use of the proper test parameters (the writable parameters in this object), the test parameters MUST be set either prior to or at the same time as (in the same SetParameterValues) setting the DiagnosticState to Requested. When requested, the CPE SHOULD wait until after completion of the communication session with the ACS before starting the diagnostic. When the diagnostic initiated by the ACS is completed (successfully or not), the CPE MUST establish a new connection to the ACS to allow the ACS to view the results, indicating the Event code "8 DIAGNOSTICS COMPLETE" in the Inform message.
Interface	string(256)	C	C	Specifies the WAN or LAN IP-layer interface over which the test is to be performed. The content is the full hierarchical parameter name of the interface. The following is a WAN interface example: "InternetGatewayDevice.WANDevice.1.-WANConnectionDevice.2.WANPPP-Connection.1" The following is a LAN interface example: "InternetGatewayDevice.LANDevice.1.LAN-HostConfigManagement.IPInterface.1"
Host	string(256)	C	C	Host name or address of the host to ping.
NumberOfRepetitions	unsignedInt [1:]	C	C	Number of repetitions of the ping test to perform before reporting the results.
Timeout	unsignedInt [1:]	C	C	Timeout in milliseconds for the ping test.
DataBlockSize	unsignedInt [1:65535]	C	C	Size of the data block in bytes to be sent for each ping.
DSCP	unsignedInt [0:64]	C	C	DiffServ codepoint to be used for the test packets. By default the CPE should set this value to zero.
SuccessCount	unsignedInt	-	C	Result parameter indicating the number of successful pings (those in which a successful response was received prior to the timeout) in the most recent ping test.
FailureCount	unsignedInt	-	C	Result parameter indicating the number of failed pings in the most recent ping test.
AverageResponseTime	unsignedInt	-	C	Result parameter indicating the average response time in milliseconds over all repetitions with successful responses of the most recent ping test. If there were no successful responses, this value MUST be zero.

Name[10]	Type	Write[11]	Read	Description
MinimumResponseTmie	unsignedInt	-	C	Result parameter indicating the minimum response time in milliseconds over all repetitions with successful responses of the most recent ping test. If there were no successful responses, this value MUST be zero.
MaximumResponseTime	unsignedInt	-	C	Result parameter indicating the maximum response time in milliseconds over all repetitions with successful responses of the most recent ping test. If there were no successful responses, this value MUST be zero.
InternetGatewayDevice.LANDevice.{i}	object	-	R	Each instance contains all LAN-related objects for a given bridged subnet.
LANEthernetInterfaceNumberOfEntries	unsignedInt	-	R	Number of instances of LANEthernetInterface-Config in this LANDevice.
LANUSBInterfaceNumberOfEntries	unsignedInt	-	R	Number of instances of LANUSBInterfaceConfig in this LANDevice.
LANWLANConfigurationNumberOfEntries	unsignedInt	-	R	Number of instances of WLANConfiguration in this LANDevice.
InternetGatewayDevice.LANDevice.{i}.LAN-HostConfigManagement.	object	-	R	This object enables reporting of LAN-related device information and setting and configuring LAN IP addressing.
DHCPServerConfigurable	boolean	R	R	Enables the configuration of the DHCP server on the LAN interface. If this variable is set to false, the CPE should restore its default DHCP server settings.
DHCPServerEnable	boolean	R	R	Enables or disables the DHCP server on the LAN interface.
DHCPRelay	boolean	-	R	Indicates if the DHCP server performs the role of a server (0) or a relay (1) on the LAN interface.
MinAddress	string	R	R	Specifies first address in the pool to be assigned by the DHCP server on the LAN interface.
MaxAddress	string	R	R	Specifies last address in the pool to be assigned by the DHCP server on the LAN interface.
ReservedAddresses	string	R	R	Comma separated list of addresses marked reserved from the address allocation pool.
SubnetMask	string	R	R	Specifies the client's network subnet mask.
DNSServers	string	R	R	Comma separated list of DNS servers offered to DHCP clients. Support for more than three DNS Servers is Optional.
DomainName	string(64)	R	R	Sets the domain name to provide to clients on the LAN interface.
IPRouters	string	R	R	Comma separated list of IP addresses of routers on this subnet. Also known as default gateway. Support for more than one Router address is Optional.
DHCPLeaseTime	int[-1:]	O	O	Specifies the lease time in seconds of client assigned addresses. A value of -1 indicates an infinite lease.

Name[10]	Type	Write[11]	Read	Description
UseAllocatedWAN	string	O	O	Enumeration of: "Normal" "UseAllocatedSubnet" "Passthrough" If Normal, then DHCP addresses are to be allocated out of a private address pool. If UseAllocatedSubnet, instructs the CPE to allocate DHCP addresses from the WAN subnet block for the WAN connection identified in AssociatedConnection. If Passthrough, then the specified LAN Host identified by the Passthrough MAC address is given the WAN IP address.
AssociatedConnection	string(256)	O	O	Specifies the connection instance for the connection to be used for address allocation if UseAllocatedWAN is set to UseAllocatedSubnet or Passthrough. The content is the full hierarchical parameter name of the default layer-3 connection object. Example: "InternetGateway-Device.WANDevice.1.WANConnectionDevice.2.-WANPPPConnection.1".
PassthroughLease	unsignedInt	O	O	DHCP lease time in seconds given to the specified LAN Host when the WAN IP address is passed-through. Note: A temporary private IP address with short lease (for example, 1 min) may be given to the passthrough LAN Host before the WAN IP address is acquired.
PassthroughMACAddress	string	O	O	Hardware address of the LAN Host that is used to passthrough the WAN IP address if UseAllocatedWAN is "Passthrough".
AllowedMACAddresses	string	O	O	Represents a comma-separated list of hardware addresses that are allowed to connect to this connection if MACAddressControlEnabled is 1 for a given interface.
IPInterfaceNumberOfEntries	unsignedInt	-	R	Number of IP interface at LAN side of the CPE. 1 is a typical value for CPE not supporting Multihomed interfaces. Support for more than one interface instance is Optional.
InternetGatewayDevice.LANDevice.{i}.LAN-HostConfigManagement.IPInterface.{i}.	object	O	R	IP interface table.
Enable	boolean	R	R	Enables or disables this entry. On creation, an entry is disabled by default.
IPInterfaceIPAddress	string	R	R	IP address of the LAN-side interface of the CPE.
IPInterfaceSubnetMask	string	R	R	Subnet mask of the LAN-side interface of the IGD.
IPInterfaceAddressingType	string	R	R	Represents the addressing method used to assign the LAN-side IP address of the CPE on this interface. Enumeration of: "DHCP" "Static" "AutoIP"
InternetGatewayDevice.LANDevice.{i}.LAN-EthernetInterfaceConfig.{i}.	object	-	C	This object models an Ethernet LAN connection on a CPE device. This object must be implemented for CPE that contain an Ethernet interface on the LAN side.
Enable	boolean	C	C	Enables or disables this interface.

Name[10]	Type	Write[11]	Read	Description
Status	string	-	C	Indicates the status of this interface. Enumeration of: “Up” “NoLink” “Error” “Disabled”
MACAddress	string	-	C	The physical address of the interface.
MACAddressControlEnabled	boolean	C	C	Indicates whether MAC Address Control is enabled or not on this interface. MAC Address Control limits the clients that connect to those that match a list of allowed MAC addresses specified in InternetGatewayDevice.LANDevice.{i}.LAN-HostConfigManagement.AllowedMACAddresses.
MaxBitRate	string	C	C	The maximum upstream and downstream bit rate available to this connection. Enumeration of: “10” “100” “1000” “Auto”
DuplexMode	string	C	C	The duplex mode available to this connection. Enumeration of: “Half” “Full” “Auto”
InternetGatewayDevice.LANDevice.{i}.LAN-EthernetInterfaceConfig.{i}.Stats.	object	-	C	This object contains statistics for an Ethernet LAN interface on a CPE device.
BytesSent	unsignedInt	-	C	Total number of bytes sent over the interface since the CPE was last reset.
BytesReceived	unsignedInt	-	C	Total number of bytes received over the interface since the CPE was last reset.
PacketsSent	unsignedInt	-	C	Total number of packets sent over the interface since the CPE was last reset.
PacketsReceived	unsignedInt	-	C	Total number of packets received over the interface since the CPE was last reset.
InternetGatewayDevice.LANDevice.{i}.LAN-USBInterfaceConfig.{i}.	object	-	C	This object models a USB LAN connection on a CPE device. This object must be implemented for CPE that contain a USB interface on the LAN side.
Enable	boolean	C	C	Enables or disables this interface.
Status	string	-	C	Indicates the status of this interface. Enumeration of: “Up” “NoLink” “Error” “Disabled”
MACAddress	string	-	C	The physical address of the interface.
MACAddressControlEnabled	boolean	C	C	Indicates whether MAC Address Control is enabled or not on this interface. MAC Address Control limits the clients that connect to those that match a list of allowed MAC addresses specified in InternetGatewayDevice.LANDevice.{i}.LAN-HostConfigManagement.AllowedMACAddresses.
Standard	string(6)	-	C	USB version supported by the device.

Name[10]	Type	Write[11]	Read	Description
Type	string	-	C	Type of the USB interface. Enumeration of: “Host” “Hub” “Device”
Rate	string	-	C	Speed of the USB interface. Enumeration of: “Low” “Full” “High” (USB 2.0)
Power	string	-	C	Power configuration of the USB interface. Enumeration of: “Self” “Bus” “Unknown”
InternetGatewayDevice.LANDevice.{i}.LAN-USBInterfaceConfig.{i}.Stats.	object	-	C	This object contains statistics for a USB LAN interface on a CPE device.
BytesSent	unsignedInt	-	C	Total number of bytes sent over the interface since the CPE was last reset.
BytesReceived	unsignedInt	-	C	Total number of bytes received over the interface since the CPE was last reset.
CellsSent	unsignedInt	-	C	Total number of cells sent over the interface since the CPE was last reset.
CellsReceived	unsignedInt	-	C	Total number of cells received over the interface since the CPE was last reset.
InternetGatewayDevice.LANDevice.{i}.WLAN-Configuration.{i}.	object	-	C	This object models an 802.11 LAN connection on a CPE device. This object must be implemented for CPE that contain an 802.11 interface on the LAN side.
Enable	boolean	C	C	Enables or disables this interface.
Status	string	-	C	Indicates the status of this interface. Enumeration of: “Up” “Error” “Disabled”
BSSID	string	-	C	The MAC address of the interface.
MaxBitRate	string(4)	C	C	The maximum upstream and downstream bit rate available to this connection in Mbps. Either “Auto”, or the largest of the OperationalDataTransmitRates values.
Channel	unsignedInt [0:255]	C	C	The current radio channel used by the connection. Note: There is currently no way of requesting automatic selection of a channel.
SSID	string(32)	C	C	The current service set identifier in use by the connection. The SSID is an identifier that is attached to packets sent over the wireless LAN that functions as a “password” for joining a particular radio network (BSS). Note: If an access point wishes to be known by more than one SSID, it must provide a WLANConfiguration instance for each SSID.

Name[10]	Type	Write[11]	Read	Description
BeaconType	string	C	C	The breacon types to be use with this connection. Enumeration of: “None” “Basic” “WPA” “11i” (Optional) “BasicandWPA” (Optional) “Basicand11i’ (Optional) “WPAand11i” (Optional) “BasicandWPAand11i” (Optional)
MACAddressControlEnabled	bool	C	C	Indicates whether MAC Address Control is enabled or not on this interface. MAC Address Control limits the clients that connect to those that match a list of allowed MAC addresses specified in InternetGatewayDevice.LANDevice.{i}.LAN-HostConfigManagement.AllowedMACAddresses.
Standard	string	-	C	Indicates which IEEE 802.11 mode the device is currently operating in. Enumeration of: “a” “b” “g”
WEPKeyIndex	unsignedInt [1:4]	C	C	The index of the default WEP key.
KeyPassphrase	string(63)	C	-	A passphrase from which the WEP keys were generated. This parameter for information only—the CPE is not responsible for generating the key based on the passphrase. This parameter is the same as the parameter InternetGatewayDevice.LANDevice.{i}.WLAN-Configuration.{i}.PreSharedKey.1.KeyPassprhase for the same instance of WLANConfiguration. When either parameter is changed, the value of the other is changed as well. This must either be a valid key length divided by 8, in which case each byte contributes 8 bits to the key, or else must consist of Hex digits and be a valid key length divided by 4, in which case each byte contributes 4 bits to the key. Note: If a passphrase is used, all four WEP keys will be the same. When read, this parameter returns an empty string, regardless of the actual value.
WEPEncryptionLevel	string(64)	-	C	Comma-separated list of the supported key lengths. Each entry in the list is an enumeration of: “Disabled” “40-bit” “104-bit” Any additional vendor-specific values must start with the key length in bits.
BasicEncryptionModes	string(31)	C	C	Encryption modes that are available when basic 802.11 is enabled. “WEPEncryption” implies that all wireless clients can use WEP for data encryption. Enumeration of: “None” “WEPEncryption”

Name[10]	Type	Write[11]	Read	Description
BasicAuthenticationMode	string(31)	C	C	Authentication modes that are available when basic 802.11 is enabled. Enumeration of: “None” “EAPAuthentication” (Optional)
WPAEncryptionModes	string(31)	C	C	Encryption modes that are available when WPA is enabled. Enumeration of: “WEPEncryption” “TKIPEncryption” “WEPandTKIPEncryption” “AESEncryption” (Optional) “WEPandAESEncryption” (Optional) “TKIPandAESEncryption” (Optional) “WEPandTKIPandAESEncryption” (Optional)
WPAAuthenticationMode	string(31)	C	C	Authentication modes that are available when WPA is enabled. Enumeration of: “PSKAuthentication” “EAPAuthentication” (Optional)
IEEE11iEncryptionModes	string(31)	O	O	Encryption modes that are available when 802.11i is enabled. Enumeration of: “WEPEncryption” “TKIPEncryption” “WEPandTKIPEncryption” “AESEncryption” (Optional) “WEPandAESEncryption” (Optional) “TKIPandAESEncryption” (Optional) “WEPandTKIPandAESEncryption” (Optional)
IEEE11iAuthenticationMode	string(31)	O	O	Authentication modes that are available when 802.11i is enabled. Enumeration of: “PSKAuthentication” “EAPAuthentication” (Optional) “EAPandPSKAuthentication” (Optional)
PossibleChannels	string (1024)	-	C	The possible radio channels for the wireless standard (a, b or g) and the regulatory domain. Comma-separated list. Ranges in the form “n-m” are permitted. For example, for 802.11b and North America, would be “1-11”.
BasicDataTransmitRates	string(256)	C	C	Comma-separated list of the maximum access point data transmit rates in Mbps for unicast, multicast and broadcast frames. For example, a value of “1,2”, indicates that unicast, multicast and broadcast frames can be transmitted at 1 Mbps and 2 Mbps.
OperationalDataTransmitRates	string(256)	C	C	Comma-separated list of the maximum access point data transmit rates in Mbps for unicast frames (a superset of BasicDataTransmitRates). Given the value of BasicDataTransmitRates from the example above, OperationalDataTransmit-Rates might be “1,2,5.5,11”, indicating that unicast frames can additionally be transmitted at 5.5 Mbps and 11 Mbps.

Name[10]	Type	Write[11]	Read	Description
PossibleDataTransmitRates	string(256)	-	C	Comma-separated list of the data transmit rates for unicast frames at which the access point will permit a station to connect (a subset of OperationalDataTransmitRates). Given the values of BasicDataTransmitRates and OperationalDataTransmitRates from the examples above, PossibleDataTransmitRates might be "1,2,5.5", indicating that the AP will only permit connections at 1 Mbps, 2 Mbps and 5.5 Mbps, even though it could theoretically accept connections at 11 Mbps.
InsecureOOBAccessEnabled	boolean	O	O	Indicates whether insecure write access via mechanisms other than the CPE WAN Management Protocol is permitted to the parameters in this object.
BeaconAdvertisementEnabled	boolean	O	O	Indicates whether or not the access point is sending out beacons.
RadioEnabled	boolean	C	C	Indicates whether or not the access point radio is enabled.
AutoRateFallBackEnabled	boolean	C	C	Indicates whether the access point can automatically reduce the data rate in the event of undue noise or contention.
LocationDescription	string (4096)	C	C	An XML description of information used to identify the access point by name and physical location. The CPE is not expected to parse this string, but simply to treat it as an opaque string. An empty string indicates no location has been set.
RegulatoryDomain	string(3)	O	O	802.11d Regulatory Domain String. First two octets are ISO/IEC 3166-1 two-character country code. The third octet is either " " (all environments), "O" (outside) or "I" (inside).
TotalPSKFailures	unsignedInt	-	O	The number of times pre-shared key (PSK) authentication has failed (relevant only to WPA and 802.11i).
TotalIntegrityFailures	unsignedInt	-	O	The number of times the MICHAEL integrity check has failed (relevant only to WPA and 802.11i)
ChannelsInUse	string (1024)	-	O	The channels that the access point determines to be currently in use (including any that it is using itself). Comma-separated list. Ranges in the form "n-m" are permitted.
DeviceOperationMode	string(31)	O	O	The current access-point operating mode. The optional modes permit the AP to be configured as a wireless bridge (to bridge two wired networks), repeater (a bridge that also serves wireless clients), or wireless station. Ad hoc stations are not supported. Enumeration of: "InfrastructureAccessPoint" "WirelessBridge" (Optional) "WirelessRepeater" (Optional) "WirelessStation" (Optional)
DistanceFromRoot	unsignedInt	O	O	The number of hops from the root access point to the wireless repeater or bridge.
PeerBSSID	string	O	O	The MAC address of the peer in wireless repeater or bridge mode.

Name[10]	Type	Write[11]	Read	Description
AuthenticationServiceMode	string	O	O	Indicates whether another service is involved in client authentication (LinkAuthentication for a co-located authentication server; RadiusClient for an external RADIUS server). Enumeration of: “None” “LinkAuthentication” (Optional) “RadiusClient” (Optional)
TotalBytesSent	unsignedInt	-	C	Total number of bytes sent over the interface since the CPE was last reset.
TotalBytesReceived	unsignedInt	-	C	Total number of bytes received over the interface since the CPE was last reset.
TotalPacketsSent	unsignedInt	-	C	Total number of packets sent over the interface since the CPE was last reset.
TotalPacketsReceived	unsignedInt	-	C	Total number of packets received over the interface since the CPE was last reset.
TotalAssociations	unsignedInt	-	C	The number of devices currently associated with the access point. This corresponds to the number of entries in the AssociatedDeivce table.
InternetGatewayDevice.LANDevice.{i}.WLAN-Configuration.{i}.AssociatedDevice.{i}.	object	-	C	A table of the devices currently associated with the access point. The size of this table is given by InternetGatewayDevice.LANDevice.{i}.WLAN-Configuration.{i}.TotalAssociations. This object must be implemented for CPE that contain an 802.11 interface on the LAN side.
AssociatedDeviceMACAddress	string	-	C	The MAC address of an associated device.
AssociatedDeviceIPAddress	string(64)	-	C	The IP address or DNS name of an associated device.
AssociatedDeviceAuthenticationState	boolean	-	C	Whether an associated device has authenticated (true) or not (false).
LastRequestedUnicastCipher	string(256)	-	O	The unicast cipher that was most recently used for a station with a specified MAC address (802.11i only).
LastRequestedMulticastCipher	string(256)	-	O	The multicast cipher that was most recently used for a station with a specified MAC address (802.11i only).
LastPMKId	string(256)	-	O	The pairwise master key (PMK) that was most recently used for a station with a specified MAC address (802.11i only).
InternetGatewayDevice.LANDevice.{i}.WLAN-Configuration.{i}.WEPKey.{i}.	object	-	C	This is a table of WEP keys. The size of this table is fixed with exactly 4 entries (with instance numbers 1 through 4). This object must be implemented for CPE that contain an 802.11 interface on the LAN side.
WEPKey	string(128)	C	-	A WEP key expressed as a hexadecimal string. The WEP encryption level is inferred from the key length, e.g. 10 characters for 40-bit encryption, or 26 characters for 104-bit encryption. When read, this parameter returns an empty string, regardless of the actual value.

Name[10]	Type	Write[11]	Read	Description
InternetGatewayDevice.LANDevice.{i}.WLAN-Configuration.{i}.PreSharedKey.{i}.	object	-	C	This is a table of preshared keys. The size of this table is fixed with exactly 10 entries (with instance numbers 1 through 10). This object must be implemented for CPE that contain an 802.11 interface on the LAN side.
PreSharedKey	string(64)	C	-	A literal WPA PSK expressed as a hexadecimal string. The first table entry contains the default PreSharedKey (InternetGatewayDevice.LAN-Device.{i}.WLANConfiguration.{i}.PreShared-Key.1.PreSharedKey). When read, this parameter returns an empty string, regardless of the actual value.
KeyPassphrase	string(63)	C	-	A passphrase from which WEP or PSK keys were generated. This parameter for information only—the CPE is not responsible for generating the key based on the passphrase. For WEP keys it must either be a valid key length divided by 8, in which case each byte contributes 8 bits to the key, or else must consist of Hex digits and be a valid key length divided by 4, in which case each byte contributes 4 bits to the key. For WPA PSK, the key is generated as specified by WPA, which uses PBKDF2 from PKCS #5: Password-based Cryptography Specification Version 2.0 (RFC2898).. Note: The 802.11i standard specifies rules for generation of WEP keys from a passphrase. When read, this parameter returns an empty string, regardless of the actual value.
AssociatedDeviceMACAddress	string	O	O	The MAC address associated with a preshared key, or an empty string if no MAC address is associated with the key.
InternetGatewayDevice.LANDevice.{i}.Hosts.	object	-	R	This object provides information about each of the hosts on the LAN, including those whose IP address was allocated by the CPE using DHCP as well as hosts with statically allocated IP addresses.
HostNumberOfEntries	unsignedInt	-	R	Number of entries in the Host table.
InternetGatewayDevice.LANDevice.{i}.Hosts.-Host.{i}.	object	-	R	Host table.
IPAddress	string	-	R	Current IP Address of the host.
AddressSource	string	-	R	Indicates whether the IP address of the host was allocated by the CPE using DHCP, was assigned to the host statically, or was assigned using automatic IP address allocation. Enumeration of: “DHCP” “Static” ”AutoIP”
LeaseTimeRemaining	int[-1:]	-	R	DHCP lease time remaining in seconds. A value of -1 indicates an infinite lease. The value must be 0 (zero) if the AddressSource is not DHCP.
MACAddress	string	-	R	MAC address of the host.
HostName	string(64)	-	R	The device’s host name or empty string if unknown.

Name[10]	Type	Write[11]	Read	Description
InterfaceType	string	-	R	Type of physical interface through which this host is connected to the CPE. Enumeration of: “Ethernet” “USB” “802.11” “HomePNA” “HomePlug” “Other”
Active	boolean	-	R	Whether or not the host is currently present on the LAN. The method of presence detection is a local matter to the CPE. The ability to list inactive hosts is Optional. If the CPE includes inactive hosts in this table, this variable MUST be set to zero for each inactive host. The length of time an inactive host remains listed in this table is a local matter to the CPE.
InternetGatewayDevice.WANDevice.{i}.	object	-	R	Each instance contains all objects associated with a particular physical WAN interface.
WANConnectionNumberOfEntries	unsignedInt	-	R	Number of instances of WANConnectionDevice in this WANDevice.
InternetGatewayDevice.WANDevice.{i}.WAN-CommonInterfaceConfig.	object	-	R	This object models WAN interface properties common across all connection instances.
EnabledForInternet	boolean	R	R	Used to enable or disable access to and from the Internet across all connection instances.
WANAccessType	string	-	R	Specifies the WAN access (modem) type. Enumeration of: “DSL” “Ethernet” “POTS”
Layer1UpstreamMaxBitRate	unsignedInt	-	R	Specifies the maximum upstream theoretical bit rate for the WAN device in bits per second.
Layer1DownstreamMaxBitRate	unsignedInt	-	R	Specifies the maximum downstream theoretical bit rate for the WAN device in bits per second.
PhysicalLinkStatus	string	-	R	Indicates the state of the physical connection (link) from WANDevice to a connected entity. Enumeration of: “Up” “Down” “Initializing” “Unavailable”
WANAccessProvider	string(256)	-	O	Name of the Service Provider providing link connectivity on the WAN.
TotalBytesSent	unsignedInt	-	R	The cumulative counter for total number of bytes sent upstream across all connection service instances on the WAN device.
TotalBytesReceived	unsignedInt	-	R	The cumulative counter for total number of bytes received downstream across all connection service instances on the WAN device.
TotalPacketsSent	unsignedInt	-	R	The cumulative counter for total number of packets (IP or PPP) sent upstream across all connection service instances on the WAN device.
TotalPacketsReceived	unsignedInt	-	R	The cumulative counter for total number of packets (IP or PPP) received downstream across all connection service instances on the WAN device.

Name[10]	Type	Write[11]	Read	Description
MaximumActiveConnections	unsignedInt	-	O	Indicates the maximum number of active connections the CPE can simultaneously support.
NumberOfActiveConnections	unsignedInt	-	O	Number of WAN connection service instances currently active on this WAN interface.
InternetGatewayDevice.WANDevice.{i}.WAN-CommonInterfaceConfig.Connection.{i}.	object	-	O	Active connection table.
ActiveConnectionDeviceContainer	string(256)	-	O	Specifies a WAN connection device object associated with this connection instance. The content is the full hierarchical parameter name of the WAN connection device. Example: "Internet-GatewayDevice.WANDevice.1.WANConnection-Device.2".
ActiveConnectionServiceID	string(256)	-	O	Specifies a WAN connection object associated with this connection instance. The content is the full hierarchical parameter name of the layer-3 connection object. Example: "InternetGateway-Device.WANDevice.1.WANConnectionDevice.2.-WANPPPConnection.1".
InternetGatewayDevice.WANDevice.{i}.WAN-DSLInterfaceConfig.	object	-	C	This object models physical layer properties specific to a single physical connection of a DSL modem used for Internet access on a CPE. This object is required for a CPE with a DSL modem WAN interface, and is exclusive of any other WAN*InterfaceConfig object within a given WAN-Device instance.
Enable	boolean	C	C	Enables or disables the link.
Status	string	-	C	Status of the DSL physical link. Enumeration of: "Up" "Initializing" "EstablishingLink" "NoSignal" "Error" "Disabled"
ModulationType	string	-	O	Indicates the type of modulation used on the connection. Enumeration of: "ADSL_G.dmt" "ADSL_G.lite" "ADSL_G.dmt.bis" "ADSL_re-adsl" "ADSL_2plus" "ADLS_four" "ADSL_ANSI_T1.413" "G.shdsl" "IDSL" "HDSL" "SDSL" "VDSL"

Name[10]	Type	Write[11]	Read	Description
LineEncoding	string	-	O	The line encoding method used in establishing the Layer 1 DSL connection between the CPE and the DSLAM. Note: Generally speaking, this variable does not change after provisioning. Enumeration of: "DMT" "CAP" "2B1Q" "43BT" "PAM" "QAM"
DataPath	string	-	O	Indicates whether the data path is fast (lower latency) or interleaved (lower error rate). Enumeration of: "Interleaved" "Fast"
InterleaveDepth	unsignedInt	-	O	ADSL Interleaved depth. This variable is only applicable if DataPath = Interleaved.
LineNumber	int[1:]	-	O	Signifies the line pair that the modem is using to connection. LineNumber = 1 is the innermost pair.
UpstreamCurrRate	unsignedInt	-	C	The current payload bandwidth (expressed in Kbps) of the upstream DSL channel.
DownstreamCurrRate	unsignedInt	-	C	The current payload bandwidth (expressed in Kbps) of the downstream DSL channel.
UpstreamMaxRate	unsignedInt	-	C	The current attainable rate (expressed in Kbps) of the upstream DSL channel.
DownstreamMaxRate	unsignedInt	-	C	The current attainable rate (expressed in Kbps) of the downstream DSL channel.
UpstreamNoiseMargin	int	-	C	The current signal-to-noise ratio (expressed in 0.1 db) of the upstream DSL connection.
DownstreamNoiseMargin	int	-	C	The current signal-to-noise ratio (expressed in 0.1 db) of the downstream DSL connection.
UpstreamAttenuation	int	-	C	The current upstream signal loss (expressed in 0.1 dB).
DownstreamAttenuation	int	-	C	The current downstream signal loss (expressed in 0.1 dB).
UpstreamPower	int	-	C	The current output power at the CPE's DSL interface (expressed in 0.1 dBmV),
DownstreamPower	int	-	C	The current received power at the CPE's DSL interface (expressed in 0.1 dBmV),
ATURVendor	string(8)	-	C	ATU-R vendor identifier as defined in G.994.1 and T1.413.
ATURCountry	unsignedInt	-	C	T.35 country code of the ATU-R vendor as defined in G.994.1.
ATURANSIStd	unsignedInt	-	O	ATU-R T1.413 Revision Number as defined in T1.413 Issue 2.
ATURANSIRev	unsignedInt	-	O	ATU-R Vendor Revision Number as defined in T1.413 Issue 2.
ATUCVendor	string(8)	-	C	ATU-C vendor identifier as defined in G.994.1 and T1.413.
ATUCCountry	unsignedInt	-	C	T.35 country code of the ATU-C vendor as defined in G.994.1.
ATUCANSIStd	unsignedInt	-	O	ATU-C T1.413 Revision Number as defined in T1.413 Issue 2.

Name[10]	Type	Write[11]	Read	Description
ATUCANSIRev	unsignedInt	-	O	ATU-C Vendor Revision Number as defined in T1.413 Issue 2.
TotalStart	unsignedInt	-	C	Number of seconds since the beginning of the period used for collection of Total statistics. Statistics SHOULD continue to be accumulated across CPE reboots, though this may not always be possible.
ShowtimeStart	unsignedInt	-	C	Number of seconds since the most recent DSL Showtime—the beginning of the period used for collection of Showtime[14] statistics.
LastShowtimeStart	unsignedInt	-	O	Number of seconds since the second most recent DSL Showtime—the beginning of the period used for collection of LastShowtime statistics. If the CPE has not retained information about the second most recent Showtime (e.g., on reboot), the start of LastShowtime statistics MAY temporarily coincide with the start of Showtime statistics.
CurrentDayStart	unsignedInt	-	O	Number of seconds since the beginning of the period used for collection of CurrentDay statistics. The CPE MAY align the beginning of each CurrentDay interval with days in the UTC time zone, but is not required to do so. Statistics SHOULD continue to be accumulated across CPE reboots, though this may not always be possible.
QuarterHourStart	unsignedInt	-	O	Number of seconds since the beginning of the period used for collection of QuarterHour statistics. The CPE MAY align the beginning of each QuarterHour interval with real-time quarter-hour intervals, but is not required to do so. Statistics SHOULD continue to be accumulated across CPE reboots, though this may not always be possible.
InternetGatewayDevice.WANDevice.{i}.WAN-DSLInterfaceConfig.Stats.	object	-	C	This object contains statistics for a WAN DSL physical interface.
InternetGatewayDevice.WANDevice.{i}.WAN-DSLInterfaceConfig.Stats.Total.	object	-	C	This object contains DSL total statistics.
ReceiveBlocks	unsignedInt	-	C	Total number of successfully received blocks.
TransmitBlocks	unsignedInt	-	C	Total number of successfully transmitted blocks.
CellDelin	unsignedInt	-	C	Total number of cell-delineation errors (total seconds with NCD or LCD failures as defined in ITU-T Rec. G.997.1).
LinkRetrain	unsignedInt	-	C	Total number of link-retrain errors (Full Initialization Count as defined in ITU-T Rec. G.997.1).
InitErrors	unsignedInt	-	C	Total number of initialization errors (LINIT failures as defined in ITU-T Rec. G.997.1).
InitTimeouts	unsignedInt	-	C	Total number of initialization timeout errors.
LossOfFraming	unsignedInt	-	C	Total number of loss-of-framing errors (LOF failures as defined in ITU-T Rec. G.997.1).
ErroredSecs	unsignedInt	-	C	Total number of errored seconds (ES-L as defined in ITU-T Rec. G.997.1).

[14] Showtime is defined as successful completion of the DSL link establishment process. The Showtime statistics are those collected since the most recent establishment of the DSL link.

Name[10]	Type	Write[11]	Read	Description
SeverelyErroredSecs	unsignedInt	-	C	Total number of severely errored seconds (SES-L as defined in ITU-T Rec. G.997.1).
FECErrors	unsignedInt	-	C	Total number of FEC errors detected (FEC-C as defined in ITU-T Rec. G.997.1).
ATUCFECErrors	unsignedInt	-	C	Total number of FEC errors detected by the ATU-C (FEC-CFE as defined in ITU-T Rec. G.997.1).
HECErrors	unsignedInt	-	C	Total number of HEC errors detected (HEC-P as defined in ITU-T Rec. G.997.1).
ATUCHECErrors	unsignedInt	-	C	Total number of HEC errors detected by the ATU-C (HEC-PFE as defined in ITU-T Rec. G.997.1).
CRCErrors	unsignedInt	-	C	Total number of CRC errors detected (CV-C as defined in ITU-T Rec. G.997.1).
ATUCCRCErrors	unsignedInt	-	C	Total number of CRC errors detected by the ATU-C (CV-CFE as defined in ITU-T Rec. G.997.1).
InternetGatewayDevice.WANDevice.{i}.WAN-DSLInterfaceConfig.Stats.Showtime.	object	-	C	This object contains DSL statistics accumulated since the most recent DSL Showtime.
ReceiveBlocks	unsignedInt	-	C	Number of successfully received blocks since the most recent DSL Showtime.
TransmitBlocks	unsignedInt	-	C	Number of successfully transmitted blocks since the most recent DSL Showtime.
CellDelin	unsignedInt	-	C	Number of cell-delineation errors since the most recent DSL Showtime (total seconds with NCD or LCD failures as defined in ITU-T Rec. G.997.1).
LinkRetrain	unsignedInt	-	C	Number of link-retrain errors since the most recent DSL Showtime (Full Initialization Count as defined in ITU-T Rec. G.997.1).
InitErrors	unsignedInt	-	C	Number of initialization errors since the most recent DSL Showtime (LINIT failures as defined in ITU-T Rec. G.997.1).
InitTimeouts	unsignedInt	-	C	Number of initialization timeout errors since the most recent DSL Showtime.
LossOfFraming	unsignedInt	-	C	Number of loss-of-framing errors since the most recent DSL Showtime (LOF failures as defined in ITU-T Rec. G.997.1).
ErroredSecs	unsignedInt	-	C	Number of errored seconds since the most recent DSL Showtime (ES-L as defined in ITU-T Rec. G.997.1).
SeverelyErroredSecs	unsignedInt	-	C	Number of severely errored seconds since the most recent DSL Showtime (SES-L as defined in ITU-T Rec. G.997.1).
FECErrors	unsignedInt	-	C	Number of FEC errors detected since the most recent DSL Showtime (FEC-C as defined in ITU-T Rec. G.997.1).
ATUCFECErrors	unsignedInt	-	C	Number of FEC errors detected by the ATU-C since the most recent DSL Showtime (FEC-CFE as defined in ITU-T Rec. G.997.1).
HECErrors	unsignedInt	-	C	Number of HEC errors detected since the most recent DSL Showtime (HEC-P as defined in ITU-T Rec. G.997.1).
ATUCHECErrors	unsignedInt	-	C	Number of HEC errors detected by the ATU-C since the most recent DSL Showtime (HEC-PFE as defined in ITU-T Rec. G.997.1).
CRCErrors	unsignedInt	-	C	Number of CRC errors detected since the most recent DSL Showtime (CV-C as defined in ITU-T Rec. G.997.1).

Name[10]	Type	Write[11]	Read	Description
ATUCCRCErrors	unsignedInt	-	C	Number of CRC errors detected by the ATU-C since the most recent DSL Showtime (CV-CFE as defined in ITU-T Rec. G.997.1).
InternetGatewayDevice.WANDevice.{i}.WAN-DSLInterfaceConfig.Stats.LastShowtime.	object	-	O	This object contains DSL statistics accumulated since the second most recent DSL Showtime.
ReceiveBlocks	unsignedInt	-	O	Number of successfully received blocks since the second most recent DSL Showtime.
TransmitBlocks	unsignedInt	-	O	Number of successfully transmitted blocks since the second most recent DSL Showtime.
CellDelin	unsignedInt	-	O	Number of cell-delineation errors since the second most recent DSL Showtime (total seconds with NCD or LCD failures as defined in ITU-T Rec. G.997.1).
LinkRetrain	unsignedInt	-	O	Number of link-retrain errors since the second most recent DSL Showtime (Full Initialization Count as defined in ITU-T Rec. G.997.1).
InitErrors	unsignedInt	-	O	Number of initialization errors since the second most recent DSL Showtime (LINIT failures as defined in ITU-T Rec. G.997.1).
InitTimeouts	unsignedInt	-	O	Number of initialization timeout errors since the second most recent DSL Showtime.
LossOfFraming	unsignedInt	-	O	Number of loss-of-framing errors since the second most recent DSL Showtime (LOF failures as defined in ITU-T Rec. G.997.1).
ErroredSecs	unsignedInt	-	O	Number of errored seconds since the second most recent DSL Showtime (ES-L as defined in ITU-T Rec. G.997.1).
SeverelyErroredSecs	unsignedInt	-	O	Number of severely errored seconds since the second most recent DSL Showtime (SES-L as defined in ITU-T Rec. G.997.1).
FECErrors	unsignedInt	-	O	Number of FEC errors detected since the second most recent DSL Showtime (FEC-C as defined in ITU-T Rec. G.997.1).
ATUCFECErrors	unsignedInt	-	O	Number of FEC errors detected by the ATU-C since the second most recent DSL Showtime (FEC-CFE as defined in ITU-T Rec. G.997.1).
HECErrors	unsignedInt	-	O	Number of HEC errors detected since the second most recent DSL Showtime (HEC-P as defined in ITU-T Rec. G.997.1).
ATUCHECErrors	unsignedInt	-	O	Number of HEC errors detected by the ATU-C since the second most recent DSL Showtime (HEC-PFE as defined in ITU-T Rec. G.997.1).
CRCErrors	unsignedInt	-	O	Number of CRC errors detected since the second most recent DSL Showtime (CV-C as defined in ITU-T Rec. G.997.1).
ATUCCRCErrors	unsignedInt	-	O	Number of CRC errors detected by the ATU-C since the second most recent DSL Showtime (CV-CFE as defined in ITU-T Rec. G.997.1).
InternetGatewayDevice.WANDevice.{i}.WAN-DSLInterfaceConfig.Stats.CurrentDay.	object	-	O	This object contains DSL statistics accumulated during the current day.
ReceiveBlocks	unsignedInt	-	O	Number of successfully received blocks during the current day.
TransmitBlocks	unsignedInt	-	O	Number of successfully transmitted blocks during the current day.
CellDelin	unsignedInt	-	O	Number of cell-delineation errors during the current day (total seconds with NCD or LCD failures as defined in ITU-T Rec. G.997.1).

Name[10]	Type	Write[11]	Read	Description
LinkRetrain	unsignedInt	-	O	Number of link-retrain errors during the current day (Full Initialization Count as defined in ITU-T Rec. G.997.1).
InitErrors	unsignedInt	-	O	Number of initialization errors during the current day (LINIT failures as defined in ITU-T Rec. G.997.1).
InitTimeouts	unsignedInt	-	O	Number of initialization timeout errors during the current day.
LossOfFraming	unsignedInt	-	O	Number of loss-of-framing errors during the current day (LOF failures as defined in ITU-T Rec. G.997.1).
ErroredSecs	unsignedInt	-	O	Number of errored seconds during the current day (ES-L as defined in ITU-T Rec. G.997.1).
SeverelyErroredSecs	unsignedInt	-	O	Number of severely errored seconds during the current day (SES-L as defined in ITU-T Rec. G.997.1).
FECErrors	unsignedInt	-	O	Number of FEC errors detected during the current day (FEC-C as defined in ITU-T Rec. G.997.1).
ATUCFECErrors	unsignedInt	-	O	Number of FEC errors detected by the ATU-C during the current day (FEC-CFE as defined in ITU-T Rec. G.997.1).
HECErrors	unsignedInt	-	O	Number of HEC errors detected during the current day (HEC-P as defined in ITU-T Rec. G.997.1).
ATUCHECErrors	unsignedInt	-	O	Number of HEC errors detected by the ATU-C during the current day (HEC-PFE as defined in ITU-T Rec. G.997.1).
CRCErrors	unsignedInt	-	O	Number of CRC errors detected during the current day (CV-C as defined in ITU-T Rec. G.997.1).
ATUCCRCErrors	unsignedInt	-	O	Number of CRC errors detected by the ATU-C during the current day (CV-CFE as defined in ITU-T Rec. G.997.1).
InternetGatewayDevice.WANDevice.{i}.WAN-DSLInterfaceConfig.Stats.QuarterHour.	object	-	O	This object contains DSL statistics accumulated during the current quarter hour.
ReceiveBlocks	unsignedInt	-	O	Number of successfully received blocks during the current quarter hour.
TransmitBlocks	unsignedInt	-	O	Number of successfully transmitted blocks during the current quarter hour.
CellDelin	unsignedInt	-	O	Number of cell-delineation errors during the current quarter hour (total seconds with NCD or LCD failures as defined in ITU-T Rec. G.997.1).
LinkRetrain	unsignedInt	-	O	Number of link-retrain errors during the current quarter hour (Full Initialization Count as defined in ITU-T Rec. G.997.1).
InitErrors	unsignedInt	-	O	Number of initialization errors during the current quarter hour (LINIT failures as defined in ITU-T Rec. G.997.1).
InitTimeouts	unsignedInt	-	O	Number of initialization timeout errors during the current quarter hour.
LossOfFraming	unsignedInt	-	O	Number of loss-of-framing errors during the current quarter hour (LOF failures as defined in ITU-T Rec. G.997.1).
ErroredSecs	unsignedInt	-	O	Number of errored seconds during the current quarter hour (ES-L as defined in ITU-T Rec. G.997.1).
SeverelyErroredSecs	unsignedInt	-	O	Number of severely errored seconds during the current quarter hour (SES-L as defined in ITU-T Rec. G.997.1).

Name[10]	Type	Write[11]	Read	Description
FECErrors	unsignedInt	-	O	Number of FEC errors detected during the current quarter hour (FEC-C as defined in ITU-T Rec. G.997.1).
ATUCFECErrors	unsignedInt	-	O	Number of FEC errors detected by the ATU-C during the current quarter hour (FEC-CFE as defined in ITU-T Rec. G.997.1).
HECErrors	unsignedInt	-	O	Number of HEC errors detected during the current quarter hour (HEC-P as defined in ITU-T Rec. G.997.1).
ATUCHECErrors	unsignedInt	-	O	Number of HEC errors detected by the ATU-C during the current quarter hour (HEC-PFE as defined in ITU-T Rec. G.997.1).
CRCErrors	unsignedInt	-	O	Number of CRC errors detected during the current quarter hour (CV-C as defined in ITU-T Rec. G.997.1).
ATUCCRCErrors	unsignedInt	-	O	Number of CRC errors detected by the ATU-C during the current quarter hour (CV-CFE as defined in ITU-T Rec. G.997.1).
InternetGatewayDevice.WANDevice.{i}.WAN-EthernetInterfaceConfig.	object	-	C	This object models physical layer properties specific to a single Ethernet physical connection used for Internet access on a CPE. This object is required for a CPE with an Ethernet WAN interface, and is exclusive of any other WAN*InterfaceConfig object within a given WAN-Device instance. Note that this object is ***not*** related to the Ethernet protocol layer sometimes used in associated with a DSL connection.
Enable	boolean	C	C	Enables or disables this interface.
Status	string	-	C	Indicates the status of this interface. Enumeration of: "Up" "NoLink" "Error" "Disabled"
MACAddress	string	-	C	The physical address of the interface.
MaxBitRate	string	C	C	The maximum upstream and downstream bit rate available to this connection. Enumeration of: "10" "100" "Auto"
DuplexMode	string	C	C	The duplex mode available to this connection. Enumeration of: "Half" "Full" "Auto"
InternetGatewayDevice.WANDevice.{i}.WAN-EthernetInterfaceConfig.Stats.	object	-	C	This object contains statistics for an Ethernet WAN interface on a CPE device.
BytesSent	unsignedInt	-	C	Total number of bytes sent over the interface since the CPE was last reset.
BytesReceived	unsignedInt	-	C	Total number of bytes received over the interface since the CPE was last reset.
PacketsSent	unsignedInt	-	C	Total number of packets sent over the interface since the CPE was last reset.
PacketsReceived	unsignedInt	-	C	Total number of packets received over the interface since the CPE was last reset.

Name[10]	Type	Write[11]	Read	Description
InternetGatewayDevice.WANDevice.{i}.WAN-DSLConnectionManagement.	object	-	C	This object is required for a CPE with a DSL modem WAN interface
ConnectionServiceNumberOfEntries	unsignedInt	-	R	Number of table entries in the ConnectionService table.
InternetGatewayDevice.WANDevice.{i}.WAN-DSLConnectionManagement.Connection-Service.{i}.	object	-	C	This table contains an entry for each connection service.
WANConnectionDevice	string(256)	-	R	Specifies a WAN connection device object associated with this connection instance. The content is the full hierarchical parameter name of the WAN connection device. Example: "Internet-GatewayDevice.WANDevice.1.WANConnection-Device.2".
WANConnectionService	string(256)	-	R	Specifies a WAN connection object associated with this connection instance. The content is the full hierarchical parameter name of the layer-3 connection object. Example: "InternetGateway-Device.WANDevice.1.WANConnectionDevice.2.-WANPPPConnection.1".
DestinationAddress	string(256)	-	R	Destination address of the WANConnectionDevice entry. One of: PVC: VPI/VCI SVC: ATM connection name SVC: ATM address
LinkType	string	-	R	Link Type of the WANConnectionDevice entry. One of Link Types as described in WANDSLLinkConfig
ConnectionType	string	-	R	Connection Type of the WANPPPConnection or WANIPConnection entry. One of PossibleConnectionTypes as described in WAN**Connection service.
Name	string(32)	-	-	User-readable name of the connection.
InternetGatewayDevice.WANDevice.{i}.WAN-DSLDiagnostics.	object	-	C	This object is required for a CPE with an ADSL2 or ADSL2+ modem WAN interface, and optional otherwise.
LoopDiagnosticsState	string	C	C	Indicates availability of diagnostic data. One of: "None" "Requested" "Complete" Value may be set to Requested to initiate the diagnostic test, which brings down the DSL connection while the test is operating. When writing, the only allowed value is Requested. When requested, the CPE SHOULD wait until after completion of the communication session with the ACS before starting the diagnostic. When the diagnostic initiated by the ACS is completed, the CPE MUST establish a new connection to the ACS to allow the ACS to view the results, indicating the corresponding reason in the Inform message.
ACTPSDds	int	-	C	Downstream actual power spectral density. Interpretation of the value is as defined in ITU-T Rec. G.997.1.
ACTPSDus	int	-	C	Upstream actual power spectral density. Interpretation of the value is as defined in ITU-T Rec. G.997.1.

Name[10]	Type	Write[11]	Read	Description
ACTATPds	int	-	C	Downstream actual aggregate transmitter power. Interpretation of the value is as defined in ITU-T Rec. G.997.1.
ACTATPus	int	-	C	Upstream actual aggregate transmitter power. Interpretation of the value is as defined in ITU-T Rec. G.997.1.
HLINSCds	int	-	C	Downstream linear representation scale. Interpretation of the value is as defined in ITU-T Rec. G.997.1.
HLINpsds	string	-	C	Downstream linear channel characteristics per subcarrier. Comma-separated list of integers. Each successive pair of integers represents the real and imaginary parts of each complex value. Maximum number of complex pairs is 256 for ADSL and ADSL2, 512 for ADSL2+. Interpretation of the value is as defined in ITU-T Rec. G.997.1.
QLNpsds	string	-	C	Downstream quiet line noise per subcarrier. Comma-separated list of integers. Maximum number of elements is 256 for ADSL and ADSL2, 512 for ADSL2+. Interpretation of the value is as defined in ITU-T Rec. G.997.1.
SNRpsds	string	-	C	Downstream SNR per subcarrier. Comma-separated list of integers. Maximum number of elements is 256 for ADSL and ADSL2, 512 for ADSL2+. Interpretation of the value is as defined in ITU-T Rec. G.997.1.
BITSpsds	string	-	C	Downstream bit allocation per subcarrier. Comma-separated list of integers. Maximum number of elements is 256 for ADSL and ADSL2, 512 for ADSL2+. Interpretation of the value is as defined in ITU-T Rec. G.997.1.
GAINSpsds	string	-	C	Downstream gain allocation per subcarrier. Comma-separated list of integers. Maximum number of elements is 256 for ADSL and ADSL2, 512 for ADSL2+. Interpretation of the value is as defined in ITU-T Rec. G.997.1.
InternetGatewayDevice.WANDevice.{i}.WAN-ConnectionDevice.{i}.	object	C	C	Each instance contains objects associated with a given WAN link. In this case of DSL, each instance corresponds to a single ATM VC. On creation of a WANConnectionDevice instance, there are initially no connection objects contained within.
WANIPConnectionNumberOfEntries	unsignedInt	-	C	Number of instances of WANIPConnection in this WANConnectionDevice.
WANPPPConnectionNumberOfEntries	unsignedInt	-	C	Number of instances of WANPPPConnection in this WANConnectionDevice.
InternetGatewayDevice.WANDevice.{i}.WAN-ConnectionDevice.{i}.WANDSLLinkConfig.	object	-	C	This object models the ATM layer properties specific to a single physical connection of a DSL modem used for Internet access on a CPE. This object is required for a CPE with a DSL modem WAN interface, and is exclusive of any other WAN*LinkConfig object within a given WAN-ConnectionDevice instance.
Enable	boolean	C	C	Enables or disables the link. On creation of a WANConnectionDevice, this object is disabled by default.

Name[10]	Type	Write[11]	Read	Description
LinkStatus	string	-	C	Status of the link. Enumeration of: "Up" "Down" "Initializing" "Unavailable"
LinkType	string	C	C	Indicates the type of DSL connection and refers to the complete stack of protocol used for this connection. Enumeration of: "EoA" (RFC2684 bridged Ethernet over ATM) "IPoA" (RFC2684 routed IP over ATM) "PPPoA" (RFC2364 PPP over ATM) "PPPoE" (RFC2516 PPP over Ethernet on RFC2684 bridged Ethernet over ATM) "CIP" (RFC1577 Classical IP over ATM) "Unconfigured"
AutoConfig	boolean	-	C	Indicates if the CPE is currently using some auto configuration mechanisms for this connection. If this variable is TRUE, all writable variables in this connection instance become read-only. Any attempt to change one of these variables SHOULD fail and an error should be returned.
ModulationType	string	-	O	Indicates the type of DSL modulation used on the interface associated with this connection (duplication from WANDSLInterfaceConfig). Enumeration of: "ADSL_G.dmt" "ADSL_G.lite" "ADSL_G.dmt.bis" "ADSL_re-adsl" "ADSL_2plus" "ADLS_four" "ADSL_ANSI_T1.413" "G.shdsl" "IDSL" "HDSL" "SDSL" "VDSL"
DestinationAddress	string(256)	C	C	Destination address of this link. One of: PVC: VPI/VCI SVC: ATM connection name SVC: ATM address
ATMEncapsulation	string	O	O	Identifies the connection encapsulation that will be used. Enumeration of "LLC" "VCMUX"

Name[10]	Type	Write[11]	Read	Description
FCSPreserved	boolean	O	O	This flag tells if a checksum should be added in the ATM payload. It does not refer to the checksum of one of the ATM cells or AALX packets. In case of LLC or VCMUX encapsulation, this ATM checksum is the FCS field described in RFC 1483. It is only applicable in the upstream direction.
VCSearchList	string	O	O	Comma separated ordered list of VPI/VCI pairs to search if a link using the DestinationAddress cannot be established. In the form: VPI1/VCI1, VPI2/VCI2, … Example: “0/35, 8/35, 1/35”
ATMAAL	string	-	O	Describes the ATM Adaptation Layer (AAL) currently in use on the PVC. Enumeration of: “AAL1” “AAL2” “AAL3” “AAL4” “AAL5”
ATMTransmittedBlocks	unsignedInt	-	C	The current count of successfully transmitted blocks.
ATMReceivedBlocks	unsignedInt	-	C	The current count of successfully received blocks.
ATMQoS	string	O	O	Describes the ATM Quality Of Service (QoS) being used on the VC. Enumeration of: “UBR” “CBR” “GFR” “VBR-nrt” “VBR-rt” “UBR+” “ABR”
ATMPeakCellRate	unsignedInt	O	O	Specifies the upstream peak cell rate in cells per second.
ATMMaximumBurstSize	unsignedInt	O	O	Specifies the upstream maximum burst size in cells.
ATMSustainableCellRate	unsignedInt	O	O	Specifies the upstream sustainable cell rate, in cells per second, used for traffic shaping.
AAL5CRCErrors	unsignedInt	-	C	Count of the AAL5 layer cyclic redundancy check errors.
ATMCRCErrors	unsignedInt	-	C	Count of the ATM layer cyclic redundancy check errors.
ATMHECErrors	unsignedInt	-	O	Count of the number of Header Error Check related errors at the ATM layer.

Name[10]	Type	Write[11]	Read	Description
InternetGatewayDevice.WANDevice.{i}.WAN-ConnectionDevice.{i}.WANATMF5Loopback-Diagnostics	object	-	O	This object is provides access to an ATM-layer F5 OAM loopback test.
DiagnosticsState	string	C	C	Indicates availability of diagnostic data. One of: "None" "Requested" "Complete" Value may be set to Requested to initiate the diagnostic test. When writing, the only allowed value is Requested. To ensure the use of the proper test parameters (the writable parameters in this object), the test parameters MUST be set either prior to or at the same time as (in the same SetParameterValues) setting the DiagnosticState to Requested. When requested, the CPE SHOULD wait until after completion of the communication session with the ACS before starting the diagnostic. When the diagnostic initiated by the ACS is completed (successfully or not), the CPE MUST establish a new connection to the ACS to allow the ACS to view the results, indicating the Event code "8 DIAGNOSTICS COMPLETE" in the Inform message.
NumberOfRepetitions	unsignedInt [1:]	C	C	Number of repetitions of the ping test to perform before reporting the results.
Timeout	unsignedInt [1:]	C	C	Timeout in milliseconds for the ping test.
SuccessCount	unsignedInt	-	C	Result parameter indicating the number of successful pings (those in which a successful response was received prior to the timeout) in the most recent ping test.
FailureCount	unsignedInt	-	C	Result parameter indicating the number of failed pings in the most recent ping test.
AverageResponseTime	unsignedInt	-	C	Result parameter indicating the average response time in milliseconds over all repetitions with successful responses of the most recent ping test. If there were no successful responses, this value MUST be zero.
MinimumResponseTime	unsignedInt	-	C	Result parameter indicating the minimum response time in milliseconds over all repetitions with successful responses of the most recent ping test. If there were no successful responses, this value MUST be zero.
MaximumResponseTime	unsignedInt	-	C	Result parameter indicating the maximum response time in milliseconds over all repetitions with successful responses of the most recent ping test. If there were no successful responses, this value MUST be zero.

Name[10]	**Type**	**Write**[11]	**Read**	**Description**
InternetGatewayDevice.WANDevice.{i}.WAN-ConnectionDevice.{i}.WANEthernetLink-Config.	object	-	C	This object models the Ethernet link layer properties specific to a single physical connection used for Internet access on a CPE. This object is required for a CPE with an Ethernet WAN interface, and is exclusive of any other WAN*Link-Config object within a given WANConnection-Device instance. Note that this object is *not* related to the Ethernet protocol layer sometimes used in associated with a DSL connection.
EthernetLinkStatus	string	-	C	Status of the Ethernet link. Enumeration of: “Up” “Down” “Unavailable”
InternetGatewayDevice.WANDevice.{i}.WAN-ConnectionDevice.{i}.WANPOTSLinkConfig.	object	-	C	This object models the POTS link layer properties specific to a single physical connection used for Internet access on a CPE. This object is required for a CPE with a POTS WAN interface, and is exclusive of any other WAN*LinkConfig object within a given WANConnectionDevice instance.
Enable	boolean	C	C	Enables or disables the link. On creation of a WANConnectionDevice, this object is disabled by default.
LinkStatus	string	-	C	Status of the link. Enumeration of: “Up” “Down” “Dialing” “Connecting” “Unavailable”
ISPPhoneNumber	string(64)	O	C	Specifies a list of strings separated by semicolon (;), each string representing a phone number to connect to a particular ISP. The digits of the phone number follow the semantics of the ITU-T E.164 specification. Delimiters such as brackets or hyphens between the digits of a phone number are to be ignored by the CPE.
ISPInfo	string(64)	O	C	Information identifying the Internet Service Provider. The format of the string is vendor specific.
LinkType	string	O	C	This variable indicates the type of POTS link used for the dialup connection. Enumeration of: “PPP_Dialup”
NumberOfRetries	unsignedInt	O	C	The number of times the CPE should attempt an Internet connection setup before returning error.
DelayBetweenRetries	unsignedInt	O	C	The number of seconds the CPE should wait between attempts to setup an Internet connection.
Fclass	string	-	O	Specifies capabilities of the POTS modem – i.e., if it handles data (0), fax (1,2,2.0), voice (8), DSVD (80). Comma separated list of the following enumeration: “0” “1” “2” “2.0” “8” “80”

Name[10]	Type	Write[11]	Read	Description
DataModulationSupported	string	-	O	The modulation standard currently being used for data. Enumeration of: "V92" "V90" "V34" "V32bis" "V32"
DataProtocol	string	-	O	The protocol standard currently being used for data transfers. Enumeration of: "V42_LAPM" "V42_MNP4" "V14" "V80"
DataCompression	string	-	O	The compression technology implemented on the modem. Enumeration of: "V42bis" "MNP5"
PlusVTRCommandSupported	boolean	-	O	Capability for full duplex operation with data and voice.
InternetGatewayDevice.WANDevice.{i}.WAN-ConnectionDevice.{i}.WANIPConnection.{i}.	object	R	R	This object enables configuration of IP connections on the WAN interface of a CPE. This object is required for all WANConnectionDevices not employing PPP addressing, and must not be active for WANConnectionDevices that do employ PPP addressing.
Enable	boolean	R	R	Enables or disables the connection instance. On creation of a WANIPConnection instance, it is initially disabled.
ConnectionStatus	string	-	R	Current status of the connection. Enumeration of: "Unconfigured" "Connecting" "Connected" "PendingDisconnect" "Disconneting" "Disconnected"
PossibleConnectionTypes	string	-	R	A comma-separated list indicating the types of connections possible for this connection instance. Each element of the list is an enumeration of: "Unconfigured" "IP_Routed" "IP_Bridged"
ConnectionType	string	R	R	Specifies the connection type of the connection instance. Enumeration of: "Unconfigured" "IP_Routed" "IP_Bridged"
Name	string(256)	R	R	User-readable name of this connection.
Uptime	unsignedInt	-	R	The time in seconds that this connection has been up.

Name[10]	Type	Write[11]	Read	Description
LastConnectionError	string	-	R	The cause of failure for the last connection setup attempt. Enumeration of: "ERROR_NONE" "ERROR_COMMAND_ABORTED" "ERROR_NOT_ENABLED_FOR_INTERNET" "ERROR_USER_DISCONNECT" "ERROR_ISP_DISCONNECT" "ERROR_IDLE_DISCONNECT" "ERROR_FORCED_DISCONNECT" "ERROR_NO_CARRIER" "ERROR_IP_CONFIGURATION" "ERROR_UNKNOWN"
AutoDisconnectTime	unsignedInt	O	O	The time in seconds since the establishment of the connection after which connection termination is automatically initiated by the CPE. This occurs irrespective of whether the connection is being used or not. A value of 0 (zero) indicates that the connection is not to be shut down automatically.
IdleDisconnectTime	unsignedInt	O	O	The time in seconds that if the connection remains idle, the CPE automatically terminates the connection. A value of 0 (zero) indicates that the connection is not to be shut down automatically.
WarnDisconnectDelay	unsignedInt	O	O	Time in seconds the Status remains in the pending disconnect state before transitioning to disconnecting state to drop the connection.
RSIPAvailable	boolean	-	R	Indicates if Realm-specific IP (RSIP) is available as a feature on the CPE.
NATEnabled	boolean	C	R	Indicates if Network Address Translation (NAT) is enabled for this connection. This parameter MUST be writable if NAT is supported by the CPE.
AddressingType	string	O	R	The method used to assign an address to the WAN side interface of the CPE for this connection. Enumeration of: "DHCP" "Static"
ExternalIPAddress	string	O	R	This is the external IP address used by NAT for this connection. This parameter is configurable only if the AddressingType is Static.
SubnetMask	string	O	R	Subnet mask of the WAN interface. This parameter is configurable only if the AddressingType is Static.
DefaultGateway	string	O	R	The IP address of the default gateway for this connection. This parameter is configurable only if the AddressingType is Static.
DNSEnabled	boolean	O	R	Whether or not the device should attempt to query a DNS server across this connection.
DNSOverrideAllowed	boolean	O	R	Whether or not a manually set, non-empty DNS address can be overridden by a DNS entry received from the WAN.
DNSServers	string	O	R	Comma separated list of DNS server IP addresses for this connection. Support for more than three DNS Servers is Optional.
MaxMTUSize	unsignedInt [1:1540]	O	O	The maximum allowed size of an Ethernet frame from LAN-side devices.
MACAddress	string	O	R	The physical address of the WANIPConnection if applicable. Configurable only if MACAddressOverride is present and true (1).

Name[10]	Type	Write[11]	Read	Description
MACAddressOverride	boolean	O	O	Whether the value of MACAddress parameter can be overridden. If false (0), the CPE's default value is used (or restored if it had previously been overridden).
ConnectionTrigger	string	R	R	Trigger used to establish the IP connection. Enumeration of: "OnDemand" "AlwaysOn" "Manual"
RouteProtocolRx	string	R	R	Defines the Rx protocol to be used. Enumeration of: "Off" "RIPv1" (Optional) "RIPv2" (Optional) "OSPF" (Optional)
PortMappingNumberOfEntries	unsignedInt	-	R	Total number of port mapping entries.
InternetGatewayDevice.WANDevice.{i}.WAN-ConnectionDevice.{i}.WANIPConnection.{i}.-PortMapping.{i}.	object	R	R	Port mapping table.
PortMappingEnabled	boolean	R	R	Enables or disables the port mapping instance. On creation, an entry is disabled by default.
PortMappingLeaseDuration	unsignedInt	R	R	Determines the time to live, in seconds, of a port-mapping lease. A value of 0 means the port mapping is static. Static port mappings do not necessarily mean persistence of these mappings across device resets or reboots.
RemoteHost	string	R	R	This parameter is the IP address of the source of inbound packets. An empty string indicates a 'wildcard' (this will be a wildcard in most cases). CPE are required only to support wildcards. When RemoteHost is a wildcard, all traffic sent to the ExternalPort on the WAN interface of the gateway is forwarded to the InternalClient on the InternalPort. When RemoteHost is specified as one external IP address, the NAT will only forward inbound packets from this RemoteHost to the InternalClient, all other packets will be dropped.
ExternalPort	unsignedInt	R	R	The external port that the NAT gateway would listen on for connection requests to a corresponding InternalPort. Inbound packets to this external port on the WAN interface should be forwarded to InternalClient on the InternalPort. A value of zero (0) represents a 'wildcard.' If this value is a wildcard, connection request on all external ports (that are not otherwise mapped) will be forwarded to InternalClient. In the wildcard case, the value(s) of InternalPort on InternalClient are ignored.
InternalPort	unsignedInt	R	R	The port on InternalClient that the gateway should forward connection requests to. A value of zero (0) is not allowed.
PortMappingProtocol	string	R	R	The protocol of the port mapping. Enumeration of: "TCP" "UDP"

Name[10]	Type	Write[11]	Read	Description
InternalClient	string	R	R	The IP address or DNS host name of an internal client (on the LAN). Support for an IP address is mandatory, while support for DNS host names is optional. This value cannot be an empty string. It must be possible to set the InternalClient to the broadcast IP address 255.255.255.255 for UDP mappings. This is to enable multiple NAT clients to use the same well-known port simultaneously.
PortMappingDescription	string(256)	R	R	User-readable description of this port mapping.
InternetGatewayDevice.WANDevice.{i}.WAN-ConnectionDevice.{i}.WANIPConnection.{i}.-Stats.	object	-	C	This object contains statistics for all connections within the same WANConnectionDevice that share a common MAC address. The contents of this object should be identical for each such connection. This object is required for all WANConnection-Devices that can support an Ethernet-layer on this interface (e.g., PPPoE, IPoE).
EthernetBytesSent	unsignedInt	-	C	Total number of bytes sent over all connections within the same WANConnectionDevice that share a common MAC address since the CPE was last reset.
EthernetBytesReceived	unsignedInt	-	C	Total number of bytes received over all connections within the same WANConnection-Device that share a common MAC address since the CPE was last reset.
EthernetPacketsSent	unsignedInt	-	C	Total number of Ethernet packets sent over all connections within the same WANConnection-Device that share a common MAC address since the CPE was last reset.
EthernetPacketsReceived	unsignedInt	-	C	Total number of Ethernet packets received over all connections within the same WANConnection-Device that share a common MAC address since the CPE was last reset.
InternetGatewayDevice.WANDevice.{i}.WAN-ConnectionDevice.{i}.WANPPPConnection.{i}.	object	R	R	This object enables configuration of PPP connections on the WAN interface of a CPE. This object is required for all WANConnectionDevices that employ PPP addressing, and must not be active for WANConnectionDevices that do not employ PPP addressing.
Enable	boolean	R	R	Enables or disables the connection instance. On creation of a WANPPPConnection instance, it is initially disabled.
ConnectionStatus	string	-	R	Current status of the connection. Enumeration of: “Unconfigured” “Connecting” “Authenticating” “Connected” “PendingDisconnect” “Disconnecting” “Disconnected”

Name[10]	Type	Write[11]	Read	Description
PossibleConnectionTypes	string	-	R	A comma-separated list indicating the types of connections possible for this connection instance. Each element of the list is an enumeration of: "Unconfigured" "IP_Routed" "DHCP_Spoofed" "PPPoE_Bridged" "PPPoE_Relay" "PPTP_Relay" "L2TP_Relay"
ConnectionType	string	R	R	Specifies the connection type of the connection instance. Enumeration of: "Unconfigured" "IP_Routed" "DHCP_Spoofed" "PPPoE_Bridged" "PPPoE_Relay" "PPTP_Relay" "L2TP_Relay"
Name	string(256)	R	R	User-readable name of this connection.
Uptime	unsignedInt	-	R	The time in seconds that this connection has been up.
LastConnectionError	string	-	R	The cause of failure for the last connection setup attempt. Enumeration of: "ERROR_NONE" "ERROR_ISP_TIME_OUT" "ERROR_COMMAND_ABORTED" "ERROR_NOT_ENABLED_FOR_INTERNET" "ERROR_BAD_PHONE_NUMBER" "ERROR_USER_DISCONNECT" "ERROR_ISP_DISCONNECT" "ERROR_IDLE_DISCONNECT" "ERROR_FORCED_DISCONNECT" "ERROR_SERVER_OUT_OF_RESOURCES" "ERROR_RESTRICTED_LOGON_HOURS" "ERROR_ACCOUNT_DISABLED" "ERROR_ACCOUNT_EXPIRED" "ERROR_PASSWORD_EXPIRED" "ERROR_AUTHENTICATION_FAILURE" "ERROR_NO_DIALTONE" "ERROR_NO_CARRIER" "ERROR_NO_ANSWER" "ERROR_LINE_BUSY" "ERROR_UNSUPPORTED_BITSPERSECOND" "ERROR_TOO_MANY_LINE_ERRORS" "ERROR_IP_CONFIGURATION" "ERROR_UNKNOWN"
AutoDisconnectTime	unsignedInt	O	O	The time in seconds since the establishment of the connection after which connection termination is automatically initiated by the CPE. This occurs irrespective of whether the connection is being used or not. A value of 0 (zero) indicates that the connection is not to be shut down automatically.

Name[10]	Type	Write[11]	Read	Description
IdleDisconnectTime	unsignedInt	O	O	The time in seconds that if the connection remains idle, the CPE automatically terminates the connection. A value of 0 (zero) indicates that the connection is not to be shut down automatically.
WarnDisconnectDelay	unsignedInt	O	O	Time in seconds the Status remains in the pending disconnect state before transitioning to disconnecting state to drop the connection.
RSIPAvailable	boolean	-	R	Indicates if Realm-specific IP (RSIP) is available as a feature on the CPE.
NATEnabled	boolean	C	R	Indicates if Network Address Translation (NAT) is enabled for this connection. This parameter MUST be writable if NAT is supported by the CPE.
Username	string(64)	R	R	Username to be used for authentication.
Password	string(64)	R	-	Password to be usef for authentication. When read, this parameter returns an empty string, regardless of the actual value.
PPPEncryptionProtocol	string	-	O	Describes the PPP encryption protocol used between the WAN device and the ISP POP. Enumeration of: “None” “MPPE”
PPPCompressionProtocol	string	-	O	Describes the PPP compression protocol used between the WAN device and the ISP POP. Enumeration of: “None” “Van Jacobsen” “STAC LZS”
PPPAuthenticationProtocol	string	-	O	Describes the PPP authentication protocol used between the WAN device and the ISP POP. Enumeration of: “PAP” “CHAP” “MS-CHAP”
ExternalIPAddress	string	-	R	This is the external IP address used by NAT for this connection.
RemoteIPAddress	string	-	O	The remote IP address for this connection.
MaxMRUSize	unsignedInt [1:1540]	O	O	The maximum allowed size of frames sent from the remote peer.
CurrentMRUSize	unsignedInt [1:1540]	-	O	The current MRU in use over this connection.
DNSEnabled	boolean	O	R	Whether or not the device should attempt to query a DNS server across this connection.
DNSOverrideAllowed	boolean	O	R	Whether or not a manually set, non-empty DNS address can be overridden by a DNS entry received from the WAN.
DNSServers	string	O	R	Comma separated list of DNS server IP addresses for this connection. Support for more than three DNS Servers is Optional.
MACAddress	string	O	R	The physical address of the WANIPConnection if applicable. Configurable only if MACAddressOverride is present and true (1).
MACAddressOverride	boolean	O	O	Whether the value of MACAddress parameter can be overridden. If false (0), the CPE’s default value is used (or restored if it had previously been overridden).

Name[10]	Type	Write[11]	Read	Description
TransportType	string	-	R	PPP transport type of the connection. Enumeration of: "PPPoA" "PPPoE" "L2TP" (for future use) "PPTP" (for future use)
PPPoEACName	string(256)	R	R	PPPoE Access Concentrator.
PPPoEServiceName	string(256)	R	R	PPPoE Service Name.
ConnectionTrigger	string	R	R	Trigger used to establish the IP connection. Enumeration of: "OnDemand" "AlwaysOn" "Manual"
RouteProtocolRx	string	R	R	Defines the Rx protocol to be used. Enumeration of: "Off" "RIPv1" (Optional) "RIPv2" (Optional) "OSPF" (Optional)
PPPLCPEcho	unsignedInt	-	O	PPP LCP Echo period in seconds.
PPPLCPEchoRetry	unsignedInt	-	O	Number of PPP LCP Echo retries within an echo period.
PortMappingNumberOfEntries	unsignedInt	-	R	Total number of port mapping entries.
InternetGatewayDevice.WANDevice.{i}.WAN-ConnectionDevice.{i}.WANPPPConnection.-{i}.PortMapping.{i}.	object	R	R	Port mapping table.
PortMappingEnabled	boolean	R	R	Enables or disables the port mapping instance. On creation, an entry is disabled by default.
PortMappingLeaseDuration	unsignedInt	R	R	Determines the time to live, in seconds, of a port-mapping lease. A value of 0 means the port mapping is static. Static port mappings do not necessarily mean persistence of these mappings across device resets or reboots.
RemoteHost	string	R	R	This parameter is the IP address of the source of inbound packets. An empty string indicates a 'wildcard' (this will be a wildcard in most cases). CPE are required only to support wildcards. When RemoteHost is a wildcard, all traffic sent to the ExternalPort on the WAN interface of the gateway is forwarded to the InternalClient on the InternalPort. When RemoteHost is specified as one external IP address, the NAT will only forward inbound packets from this RemoteHost to the InternalClient, all other packets will be dropped.
ExternalPort	unsignedInt	R	R	The external port that the NAT gateway would listen on for connection requests to a corresponding InternalPort. Inbound packets to this external port on the WAN interface should be forwarded to InternalClient on the InternalPort. A value of zero (0) represents a 'wildcard.' If this value is a wildcard, connection request on all external ports (that are not otherwise mapped) will be forwarded to InternalClient. In the wildcard case, the value(s) of InternalPort on InternalClient are ignored.

Name[10]	Type	Write[11]	Read	Description
InternalPort	unsignedInt	R	R	The port on InternalClient that the gateway should forward connection requests to. A value of zero (0) is not allowed.
PortMappingProtocol	string	R	R	The protocol of the port mapping. Enumeration of: "TCP" "UDP"
InternalClient	string	R	R	The IP address or DNS host name of an internal client (on the LAN). Support for an IP address is mandatory, while support for DNS host names is optional. This value cannot be an empty string. It must be possible to set the InternalClient to the broadcast IP address 255.255.255.255 for UDP mappings. This is to enable multiple NAT clients to use the same well-known port simultaneously.
PortMappingDescription	string(256)	R	R	User-readable description of this port mapping.
InternetGatewayDevice.WANDevice.{i}.WAN-ConnectionDevice.{i}.WANPPPConnection.-{i}.Stats.	object	-	C	This object contains statistics for all connections within the same WANConnectionDevice that share a common MAC address. The contents of this object should be identical for each such connection. This object is required for all WANConnection-Devices that can support an Ethernet-layer on this interface (e.g., PPPoE, IPoE).
EthernetBytesSent	unsignedInt	-	C	Total number of bytes sent over all connections within the same WANConnectionDevice that share a common MAC address since the CPE was last reset.
EthernetBytesReceived	unsignedInt	-	C	Total number of bytes received over all connections within the same WANConnection-Device that share a common MAC address since the CPE was last reset.
EthernetPacketsSent	unsignedInt	-	C	Total number of Ethernet packets sent over all connections within the same WANConnection-Device that share a common MAC address since the CPE was last reset.
EthernetPacketsReceived	unsignedInt	-	C	Total number of Ethernet packets received over all connections within the same WANConnection-Device that share a common MAC address since the CPE was last reset.

Appendix 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 may specify various characteristics that determine under what conditions that Option persists.

C.2 Control of Options Using Vouchers

An Option may be disabled, enabled, or enabled with expiration. An Option that is enabled with no expiration stays enabled until the Server 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 may 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 may contain instructions to enable or disable one or more Options, MUST be digitally signed using the XML-Signature format [13]. 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 Appendix A of this specification:

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

C.3 Voucher Definition

The RPC method SetVouchers allows a Server 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 [13]. 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 Appendix B.

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>2Wire</Manufacturer>
      <OUI>00D09E</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@2wire.com,CN=2Wire,OU=CMS,O=2Wire,L=San\20Jose,
                              ST=California,C=US
        </X509IssuerName>
        <X509SerialNumber>4</X509SerialNumber>
      </X509IssuerSerial>
      <X509SubjectName>
        CN=eng.bba.certs.2wire.com,OU=CMS,O=2Wire,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>2Wire</Manufacturer>
        <OUI>00D09E</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>2Wire</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>
```

Appendix 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 Appendix A, which may 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 may be used to associate the method call with a particular web browser session.

The response to the Kicked method allows the Server to specify a URL to which the browser should be redirected. This URL may contain CGI arguments that allow the Server 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 may occur (under normal conditions):

1. The user's browser accesses a web page that requires knowledge of, or communication with, the user's CPE.

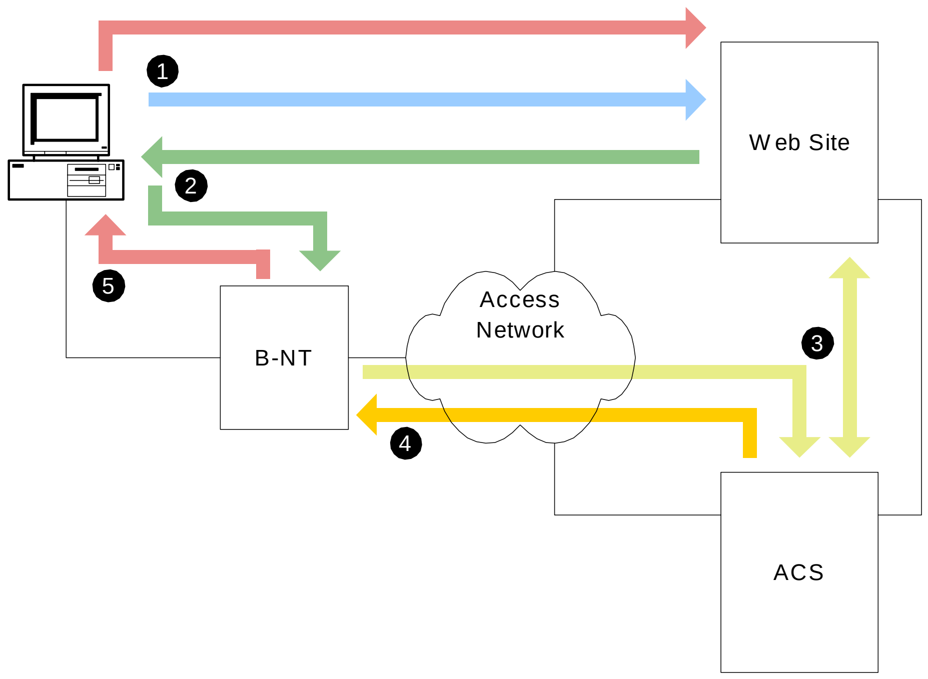
2. The web site redirects the browser to a specific URL accessible only from the CPE's private-network (LAN) interface through which the browser "kicks" the CPE, providing the CPE via CGI arguments with information it needs to follow the subsequent steps (see section D.4).
3. The CPE notifies the ACS that it has been kicked, using the "Kicked" RPC method call defined in Appendix 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 Appendix 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 may 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 Appendix 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.

Appendix E. Signed Package Format

E.1 Introduction

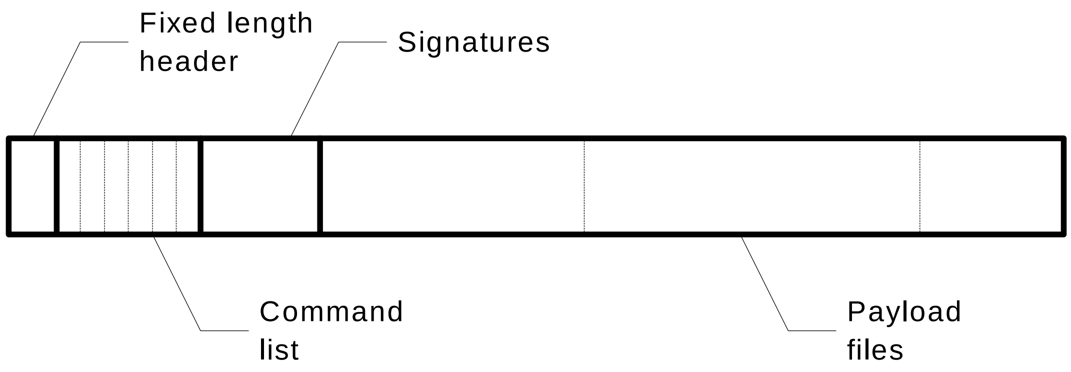
This document specifies a signed package format that may be used to securely download files into a recipient device. The format allows one or more files to be encapsulated within a single signed package. The package format allows the recipient to authenticate the source, and contains instructions for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

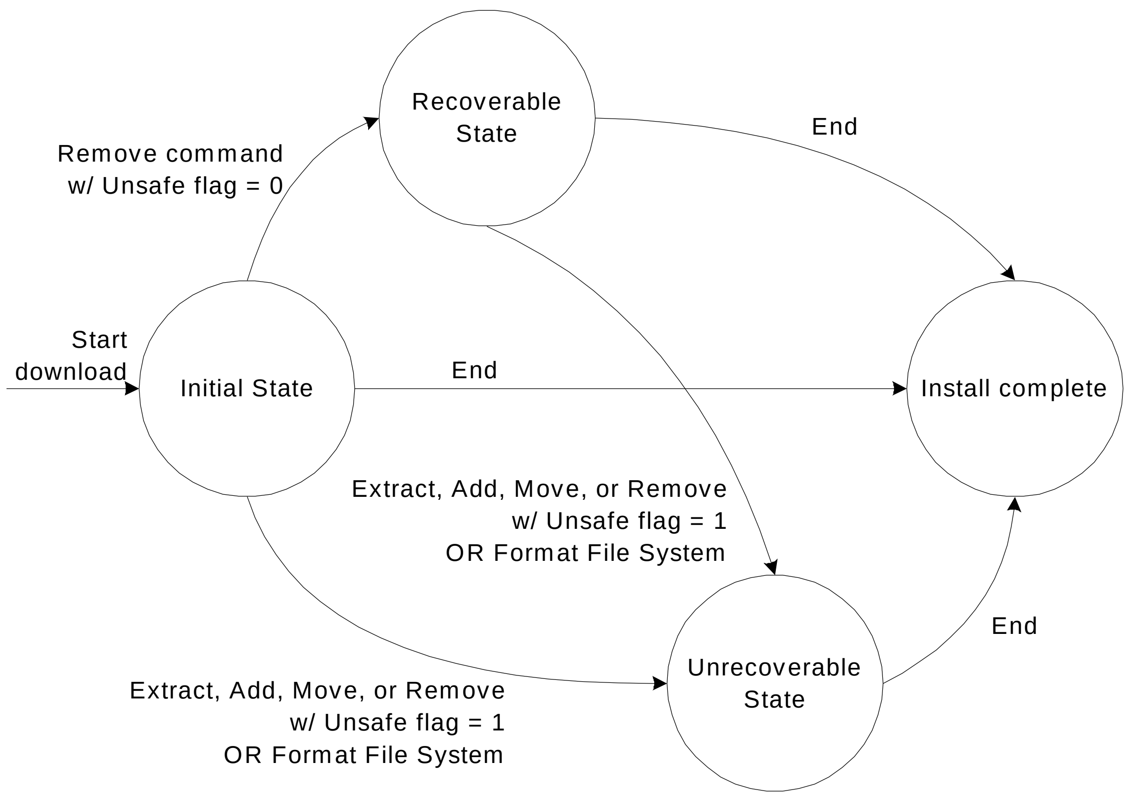
Each timeout command includes information in the Value portion of the command. The format of this information is defined in Table 72.

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 may 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 [14].

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 should 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.