TECHNICAL REPORT

TR-106
Data Model Template for TR-069-Enabled Devices

Issue: 1 Amendment 6
Issue Date: July 2011

Notice

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

This Broadband Forum Technical Report is provided AS IS, WITH ALL FAULTS. ANY PERSON HOLDING A COPYRIGHT IN THIS BROADBAND FORUM TECHNICAL REPORT, OR ANY PORTION THEREOF, DISCLAIMS TO THE FULLEST EXTENT PERMITTED BY LAW ANY REPRESENTATION OR WARRANTY, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY WARRANTY:

(A) OF ACCURACY, COMPLETENESS, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE;
(B) THAT THE CONTENTS OF THIS BROADBAND FORUM TECHNICAL REPORT ARE SUITABLE FOR ANY PURPOSE, EVEN IF THAT PURPOSE IS KNOWN TO THE COPYRIGHT HOLDER;
(C) THAT THE IMPLEMENTATION OF THE CONTENTS OF THE TECHNICAL REPORT WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

By using this Broadband Forum Technical Report, users acknowledge that implementation may require licenses to patents. The Broadband Forum encourages but does not require its members to identify such patents. For a list of declarations made by Broadband Forum member companies, please see http://www.broadband-forum.org. No assurance is given that licenses to patents necessary to implement this Technical Report will be available for license at all or on reasonable and non-discriminatory terms.

ANY PERSON HOLDING A COPYRIGHT IN THIS BROADBAND FORUM TECHNICAL REPORT, OR ANY PORTION THEREOF, DISCLAIMS TO THE FULLEST EXTENT PERMITTED BY LAW (A) ANY LIABILITY (INCLUDING DIRECT, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES UNDER ANY LEGAL THEORY) ARISING FROM OR RELATED TO THE USE OF OR RELIANCE UPON THIS TECHNICAL REPORT; AND (B) ANY OBLIGATION TO UPDATE OR CORRECT THIS TECHNICAL REPORT.

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

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

Issue History

Issue Number	Issue Date	Issue Editor	Changes
Issue 1	September 2005	Jeff Bernstein, 2Wire Christele Bouchat, Alcatel Tim Spets, Westell	Original
Issue 1 Amendment 1	November 2006	Jeff Bernstein, 2Wire John Blackford, 2Wire Mike Digdon, SupportSoft Heather Kirksey, Motive William Lupton, 2Wire Anton Okmianski, Cisco	Clarification of original document
Issue 1 Amendment 2	November 2008	William Lupton, 2Wire Håkan Westin, Tilgin	Addition of data model definition XML Schema and normative XML common object and component definitions
Issue 1 Amendment 3	September 2009	William Lupton, 2Wire Håkan Westin, Tilgin	Addition of device type XML Schema
Issue 1 Amendment 4	February 2010	William Lupton, 2Wire Paul Sigurdson, Broadband Forum	Moved data model definitions to TR-181 Issue 1
Issue 1 Amendment 5	November 2010	Paul Sigurdson, Broadband Forum	Replaced definitions of named data types such as IPAddress with references to normative XML. Minor changes to DM Schema (v1.1) and DT Schema (v1.3).
Issue 1 Amendment 6	July 2011	Sarah Banks, Cisco Andrea Colmegna, FASTWEB Tim Spets, Motorola Mobility	Removed definition of proxying, now defined in TR-069 [2]. Removed Common objects. Alias Parameter Requirements added.

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

Editors	Sarah Banks	Cisco
	Andrea Colmegna	FASTWEB
	Tim Spets	Motorola Mobility
Editors:	Sarah Banks	Cisco
Object Addressing	Andrea Colmegna	FASTWEB
Extensions	Ping Fang	Huawei Technologies
	Nils Magnusson	TeliaSonera
	Anton Okmyanskiy	Cisco
	Staffan Ungsgard	TeliaSonera
Editors:	John Blackford	Pace
Remote Management of	Ping Fang	Huawei Technologies
non-TR-069 devices	Tim Spets	Motorola Mobility
BroadbandHome™	Greg Bathrick	PMC-Sierra
Working Group Chairs	Heather Kirksey	Alcatel-Lucent
Vice Chair	Jason Walls	UNH
Chief Editor	Michael Hanrahan	Huawei Technologies

Table of Contents

List of Figures

List of Tables

Executive Summary

TR-106 specifies data model guidelines to be followed by all TR-069-enabled [2] devices. These guidelines include structural requirements for the data hierarchy, requirements for versioning of data models, and requirements for defining profiles.

In addition, TR-106 defines an XML Schema that as far as possible embodies these guidelines, and which is used for defining all TR-069 data models. This makes data model definitions rigorous, and helps to reduce the danger that different implementations will interpret data model definitions in different ways.

TR-106 also defines an XML Schema that allows a device to describe its supported TR-069 data models. This description is both specific and detailed, allowing an ACS to know exactly what is supported by the device, including any vendor-specific objects and parameters. Use of this Schema enhances interoperability and significantly eases the integration of new devices with an ACS.

1 Introduction

TR-069 [2] defines the generic requirements of the CPE WAN Management Protocol (CWMP) methods which can be applied to any TR-069-enabled CPE. It is intended to support a variety of different 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

The ability to manage the home network remotely has a number of benefits including reducing the costs associated with activation and support of broadband services, improving time-to-market for new products and services, and improving the user experience.

The following figure places TR-069 in the end-to-end management architecture:

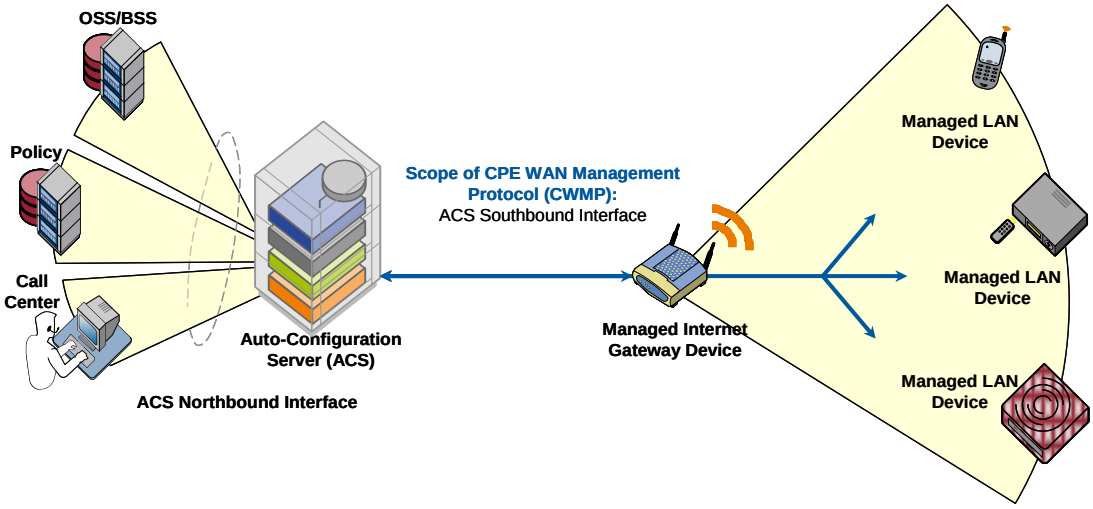
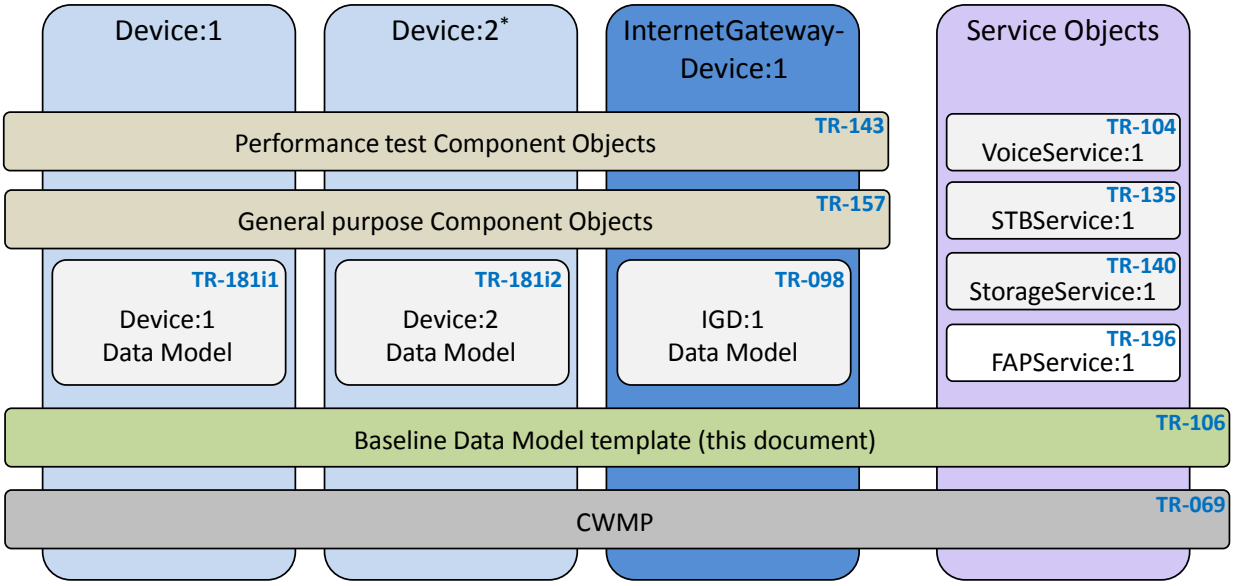


Figure 1 – Positioning in the End-to-End Architecture

The ACS is a server that resides in the network and manages devices in the subscriber premises. It uses the methods, or RPCs, defined in TR-069 to get and set the state of the device, initiate diagnostic tests, download and upload files, and manage events. Some portions of this state are common across managed devices and some are relevant only to certain device types or services.

Specification Structure

Figure 2 shows the overall specification structure for the TR-069 [2] family of standards (as currently defined). Please note that this will gradually become out of date as new documents are published.



* The Device:2 Data Model applies to all types of device, including Internet Gateway Devices (it includes everything that is in the IGD:1 data model)

Figure 2 – Specification Structure

TR-069 [2] defines the generic requirements of the CWMP methods which can be applied to any TR-069-enabled device. TR-106 (this document) specifies a baseline object structure to be supported by any TR-069-enabled device. It specifies how to structure and define data models, which are collections of objects and parameters on which the generic methods act to configure, diagnose, and monitor the state of specific devices and services. The actual data models are defined in their own specifications.

For a particular type of device, it is expected that the baseline defined in a document such as TR-181 [13] would be augmented with additional objects and parameters specific to the device type. The data model used in any TR-069-enabled device MUST follow the guidelines described in this document. These guidelines include the following aspects:

- Structural requirements for the data hierarchy
- Requirements for versioning of data models
- Requirements for defining profiles

In addition, this document defines two XML Schemas:

- An XML Schema that as far as possible embodies these guidelines, and which is used for defining all TR-069 data models. This makes data model definitions rigorous, and helps to reduce the danger that different implementations will interpret data model definitions in different ways.
- An XML Schema that allows a device to describe its supported TR-069 data models. This description is both specific and detailed, allowing an ACS to know exactly what is supported by the device, including any vendor-specific objects and parameters. Use of this Schema enhances interoperability and significantly eases the integration of new devices with an ACS.

1.1 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.
BBF	The Broadband Forum.
Base Supported Data Model	The *Data Model* that is supported by all *CPE* of a given make, model and firmware version. This refers to the *Objects* and/or *Parameters* that have code support in the current firmware.
CPE	Customer Premises Equipment; refers to any TR-069-enabled [2] device and therefore covers both Internet Gateway devices and LAN-side end devices.
Current Supported Data Model	The *Data Model* that is currently supported by an individual *CPE*, i.e. the *Base Supported Data Model* plus any additional *Objects* and/or *Parameters* supported by extra modules that have been installed on the *CPE*. This refers to the *Objects* and/or *Parameters* that have code support in the *CPE*.
Component	A named collection of *Objects* and/or *Parameters* and/or *Profiles* that can be included anywhere within a *Data Model*.
CWMP	*CPE* WAN Management Protocol. Defined in TR-069 [2], CWMP is a communication protocol between an *ACS* and *CPE* that defines a mechanism for secure auto-configuration of a *CPE* and other *CPE* management functions in a common framework.
Data Model	A hierarchical set of *Objects* and/or *Parameters* that define the managed *Objects* accessible via TR-069 [2] for a particular *CPE*.
Device	Used here as a synonym for *CPE*.
DM Instance	Data Model Schema instance document. This is an XML document that conforms to the *DM Schema* and to any additional rules specified in or referenced by the *DM Schema*.
DM Schema	Data Model Schema. This is the XML Schema [11] that is used for defining data models for use with *CWMP*.
DT Instance	Device Type Schema instance document. This is an XML document that conforms to the *DT Schema* and to any additional rules specified in or referenced by the *DT Schema*.
DT Schema	Device Type Schema. This is the XML Schema [11] that is used for describing a *Device*'s *Supported Data Model*.
Event	An indication that something of interest has happened that requires the *CPE* to notify the *ACS*.
Instantiated Data Model	The *Data Model* that currently exists on an individual *CPE*. This refers to the *Object* instances and/or *Parameters* that currently exist within the data model. It can be thought of as the *Current Supported Data Model* with all the "{i}" placeholders expanded to be the actual *Instance Numbers*. For example, "Device.Services.ABCService.{i}" in the *Current Supported Data Model* might correspond to "Device.Services.ABCService.1" and "Device.Services.ABCService.2" in the *Instantiated Data Model*.
Instance Alias	A writeable string that uniquely identifies an instance within a *Multi-Instance Object*
Instance Identifier	A value that uniquely identifies an instance within a *Multi-Instance Object*. It is either an *Instance Number* or an *Instance Alias*.
Instance Number	A read-only positive integer (>=1) that uniquely identifies an instance within a *Multi-Instance Object.*

Multi-Instance Object	An *Object* that can have multiple instances, all of which have the same structure and are located at the same level within the name hierarchy. Each instance is identified by an *Instance Identifier*.
Object	An internal node in the name hierarchy, i.e., a node that can have Object or *Parameter* children. An *Object* name is a *Path Name*.
Parameter	A name-value pair that represents part of a *CPE*'s configuration or status. A Parameter name is a *Path Name*.
Path Name	A name that has a hierarchical structure similar to files in a directory, with each level separated by a "." (dot). References an *Object* or a *Parameter*.
RPC	Remote Procedure Call.
Profile	A named collection of requirements relating to a given *Root Object*, *Service Object* or *Component*.
Root Object	The top-level *Object* of a *CPE*'s *Data Model* that contains all of the manageable *Objects*. The name of the Root *Object* is either "Device" or "InternetGatewayDevice" – the latter is used only for the TR-098 [3] InternetGatewayDevice:1 *Data Model*.
Service Object	The top-most *Object* associated with a specific service within which all *Objects* and *Parameters* associated with the service are contained.
Supported Data Model	Refers to either *Base Supported Data Model* or *Current Supported Data Model*, depending on the context.
URI	Uniform Resource Identifier [7].
URL	Uniform Resource Locator [7].

1.2 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 Data Hierarchy

The data model for a TR-069-enabled [2] device will follow a common set of structural requirements. The detailed structure depends on the nature of the device.

A device will always have a single Root Object, which will be called either "Device" or "InternetGatewayDevice". The latter is used only for the TR-098 [3] InternetGatewayDevice:1 data model.

The Root Object contains three types of sub-elements: the Objects defined in TR-181 [13] (applicable only to the "Device" Root Object) or TR-098 [3] (applicable only to the "InternetGatewayDevice" Root Object), Components defined in other specifications such as TR-143 [10] and TR-157 [12] (applicable to both the "Device" and "InternetGatewayDevice" Root Objects), and a single "Services" object that contains all Service Objects associated with specific services.

A single device might include more than one Service Object. For example, a device that serves both as a VoIP endpoint and a game device, might include both VoIP-specific and game-specific Service Objects.

A single device might also include more than one instance of the same type of Service Object. An example of when this might be appropriate is a TR-069-enabled device that provides an instance of the TR-140 [15] StorageService for each of its attached disks.

2.1.1 Data Hierarchy Requirements

A device that implements the TR-181 [13] Device:1 or Device:2 data model MUST adhere to the following structural requirements:

1) The data model MUST contain exactly one Root Object, called "Device".
2) If (and only if) the Root Object major version (Section 2.2) is 1, the Root Object MUST contain the (DEPRECATED) "DeviceSummary" parameter as specified in Section 3.7.
3) The Root Object MAY contain any of the Objects defined in TR-181 [13], and Components defined in other specifications, e.g. TR-143 [10] or TR-157 [12], with the proviso that a Component that is defined as a child of another Object can only be included if the parent Object is also included.
4) The Root Object MUST contain exactly one "Services" object.
5) The "Services" object MUST contain all of the Service Objects supported by the device. Each Service Object contains all of the objects and parameters for a particular service.
6) The "Services" object MAY contain more than one Service Object, each corresponding to a distinct service type.
7) The "Services" object MAY contain more than one instance of a Service Object of the same type.
8) Each Service Object instance MUST be appended with an Instance Number (assigned by the CPE) to allow for the possibility of multiple instances of each. For example, if the device supports the Service Object ABCService, the first instance of this Service Object might be "ABCService.1".
9) For each supported type of Service Object, a corresponding parameter in the "Services" object MUST indicate the number of instances of that Service Object type. If a particular Service Object type is supported by the device but there are currently no instances present, this parameter MUST still be present with a value of zero. The name of this parameter MUST be the name of the Service Object concatenated with "NumberOfEntries". For example, for a device that contains instances of ABCService, there MUST be a corresponding parameter in the "Services" object called "ABCServiceNumberOfEntries".

A device that implements the TR-098 [3] InternetGatewayDevice:1 data model MUST adhere to the above requirements with the following exceptions:

1) The data model MUST contain exactly one Root Object, called "InternetGatewayDevice".
2) The (DEPRECATED) "DeviceSummary" parameter MAY be absent only in an Internet Gateway Device that supports the InternetGatewayDevice version 1.0 data model, as defined in Section 2.4.2/TR-098 [3], and no Service Objects.[1]
3) The Root Object MAY contain any of the objects specific to an Internet Gateway Device as defined in TR-098 [3], and any Components defined in other specifications, e.g. TR-143 [10] or TR-157 [12], with the proviso that a Component that is defined as a child of another Object can only be included if an Internet Gateway Device object with the same name as the parent Object is also included.
4) The "Services" object MAY be absent if the device supports no Service Objects.

[1] The implication of this requirement is that a TR-098 [3] Internet Gateway Device that supports one or more Service Objects (for example, the VoiceService object defined in TR-104) is REQUIRED to support version 1.1 or greater of the InternetGatewayDevice Root Object.

Formally, the top level of the data hierarchy is defined as follows:

```
Element = Root
        | Root ".DeviceSummary"
        | Root ".Services." ServiceObject "." Instance
        | Root ".Services." ServiceObject "NumberOfEntries"
        | Root "." ComponentObject
        | TR-181DeviceRoot "." TR-181DeviceObject
        | TR-181DeviceRoot "." TR-181DeviceObject "." ComponentObject
        | TR-098GatewayRoot "." TR-098GatewayObject
        | TR-098GatewayRoot "." TR-098GatewayObject "." ComponentObject

Root = TR-181DeviceRoot
     | TR-098GatewayRoot

TR-181DeviceRoot = "Device"

TR-098GatewayRoot = "InternetGatewayDevice"

TR-181DeviceObject = // As defined in TR-181 [13], e.g. "UserInterface" or "ManagementServer"

TR-098GatewayObject = // As defined in TR-098 [3]

ServiceObject = // As defined in other specs, e.g. TR-140 [15]

ComponentObject = // As defined in other specs, e.g. TR-143 [10] or TR-157 [12]

Instance = NONZERODIGIT [DIGIT]*
```

2.1.2 Data Hierarchy Examples

Below are some examples of data hierarchies for various types of devices. (Objects are shown in bold text, parameters are shown in plain text.)

Simple device supporting the native ABCService Service Object:

Device
- DeviceSummary[2]
- **DeviceInfo**
- **ManagementServer**
- **Services**
  - ABCServiceNumberOfEntries = 1
  - **ABCService.1**
    - **ABCServiceSpecificObjects**

Device supporting both native ABCService and XYZService Service Objects:

Device
- DeviceSummary[2]
- **DeviceInfo**
- **ManagementServer**
- **Time**
- **UserInterface**
- **Services**
  - ABCServiceNumberOfEntries = 1
  - **ABCService.1**
    - **ABCServiceSpecificObjects**
  - XYZServiceNumberOfEntries = 1
  - **XYZService.1**
    - **XYZServiceSpecificObjects**

[2] The (DEPRECATED) DeviceSummary parameter applies only to major version 1 of the "Device" Root Object.

TR-098 [3] Internet Gateway Device that also supports the native ABCService and XYZService Service Objects:

InternetGatewayDevice
DeviceSummary
DeviceInfo
ManagementServer
Time
UserInterface
Layer3Forwarding
LANDeviceNumberOfEntries = 1
LANDevice.1
WANDeviceNumberOfEntries = 1
WANDevice.1
Services
ABCServiceNumberOfEntries = 1
ABCService.1
ABCServiceSpecificObjects
XYZServiceNumberOfEntries = 1
XYZService.1
XYZServiceSpecificObjects

Device supporting the native ABCService Service Object and proxying[3] for two Devices supporting the functionality of the XYZService Objects:

Device
DeviceSummary[2]
DeviceInfo
ManagementServer
EmbeddedDeviceNumberOfEntries = 2
EmbeddedDevice.1
Reference = Device.Services.XYZService.1
EmbeddedDevice.2
Reference = Device.Services.XYZService.2
Time
UserInterface
Services
ABCServiceNumberOfEntries = 1
ABCService.1
ABCServiceSpecificObjects
XYZServiceNumberOfEntries = 2
XYZService.1
XYZServiceSpecificObjects
XYZService.2
XYZServiceSpecificObjects

2.1.3 The Supported Data Model and the Instantiated Data Model

There is a distinction between a TR-069-enabled [2] device's Supported Data Model and its Instantiated Data Model.

- The Supported Data Model is those Objects and/or Parameters that have code support in the CPE.
- The Instantiated Data Model is those Object instances and/or Parameters that currently exist.

[3] Previous versions of this document described the use of Service Object instances to represent proxy-managed devices. This mechanism has been replaced by a more general Annex J /TR-069 [2] Embedded Object Mechanism, which allows any Object instance (including a Service Object instance) to represent a proxy-managed device.

TR-157 [12] defines a SupportedDataModel Object that allows a TR-069-enabled device to indicate its Supported Data Model to the ACS, which assists the ACS in managing that device.

The SupportedDataModel object has the following properties:

1) It contains a list of URLs, each of which allows the ACS to determine details of part of the Supported Data Model.

2) When the Supported Data Model changes, e.g. because software is loaded or unloaded, entries are added to or removed from this list of URLs.

2.2 Object Versioning

To allow the definition of a Service Object or Root Object to change over time, the definition of a Service Object or Root Object MUST have an explicitly specified version.

Version numbering of Service Objects and Root Objects is defined to use a major/minor version numbering convention. The object version is defined as a pair of integers, where one integer represents the major version, and the second integer represents the minor version. The version MUST be written with the two integers separated by a dot (Major.Minor).

The first version of a given object SHOULD be defined as version "1.0".

For each subsequent version of the object, if the later version is compatible with the previous version, then the major version SHOULD remain unchanged, and the minor version SHOULD be incremented by one. For example, the next compatible version after "2.17" would be "2.18". The requirements for a version to be considered compatible with an earlier version are described in Section 2.2.1.

For each subsequent version of the object, if the later version is not compatible with the previous version, then the major version MUST increment by one, and the minor version MAY reset back to zero. For example, the next incompatible version after "2.17" might be "3.0".

2.2.1 Requirements for Compatible Versions

For one version of an object to be considered compatible with another version, the later version MUST be a strict superset of the earlier version. Using major/minor versioning, this requirement applies only between minor versions that share the same major version.

More specifically, this requires the following of the later version with respect to all earlier versions to which it is to be compatible:

- The later version MAY add objects and parameters not previously in any earlier version, but MUST NOT remove objects or parameters already defined in earlier versions.
- The later version MUST NOT modify the definition of any parameter or object already defined in an earlier version (unless the original definition was clearly in error and has to be modified as an erratum or clarified through a corrigendum process).
- The later version MUST NOT require any of the objects or parameters that have been added since the earliest compatible version to be explicitly operated upon by the ACS to ensure proper operation of the device (except those functions specifically associated with functionality added in later versions). That is, the later version will accommodate an ACS that knows nothing of elements added in later versions.

The goal of the above definition of compatibility is intended to ensure bi-directional compatibility between an ACS and CPE. Specifically that:

- If an ACS supports only an earlier version of an object as compared to the version supported by the CPE, the ACS can successfully manage that object in the CPE as if it were the earlier version.
- If a CPE supports only an earlier version of an object as compared to the version supported by an ACS, the ACS can successfully manage that object in the CPE as if it were the later version (without support for new components defined only in later versions).

2.2.2 Version Notation

For objects, the following notation is defined to identify specific versions:

Notation	Description	Example
ObjectName:Major.Minor	Refers to a specific version of the object.	Device:1.0
ObjectName:Major	Refers to any minor version of the object with the specified major version.	Device:1
ObjectName	Refers to any version of the object.	Device

Note that the version notation defined here is *only* to be used for purposes of documentation and in the content of the DeviceSummary parameter defined in Section 3.7. The actual names of objects and parameters in the data model MUST NOT include version numbers.

2.3 Profiles

Note: Originally, profiles were seen as a means of limiting the variability that an ACS needs to accommodate among various devices that it might manage. This feature is now provided by the TR-157 [12] SupportedDataModel object (see Section 2.1.3) and associated Device Type XML documents (DT Instances).

A profile is a named collection of requirements associated with a given object. A device can adhere to zero or more profiles. Adherence to a profile means that the device supports all of the requirements defined by that profile. The use of profiles gives Service Providers a shorthand means of specifying CPE data model support requirements.

The following sections define the conventions to be used when defining profiles associated with TR-069 [2] data models.

2.3.1 Scope of Profiles

A given profile is defined only in the context of a specific Service Object or Root Object with a specific major version. For each profile definition, the specific object name and major version to which the profile is to apply MUST be explicitly identified.

A profile's name MUST be unique among profiles defined for the same object and major version, but a name MAY be reused to define a different profile for a distinct combination of object name and major version. For example, if we define profile "A" associated with object "X:2" (major version 2 of object X), the same name "A" might be used to define a different profile for object "Y:1" or for object "X:3".

A given profile is defined in association with a minimum minor version of a given object. The minimum REQUIRED version of an object is the minimum version that includes all of the REQUIRED elements defined by the profile. For each profile definition, the specific minimum version MUST be explicitly identified.

2.3.2 Multiple Profile Support

For a given type of Service Object or Root Object, multiple profiles MAY be defined. Profiles MAY be defined that have either independent or overlapping requirements.

To maximize interoperability, a device that fully implements the (DEPRECATED) DeviceSummary parameter (Section 3.7) MUST indicate all profiles that it supports. That is, it has to indicate all profiles whose definition is a subset of the support provided by that device. Doing so maximizes the likelihood that an ACS will be aware of the definition of the indicated profiles. For example, if profile "A" is a subset of profile "B", and a device supports both, by indicating support for both "A" and "B" an ACS that is unaware of profile "B" will at least recognize the device's support for profile "A".

2.3.3 Profile Versions

To allow the definition of a profile to change over time, the definition of every profile MUST have an associated version number.

Version numbering of profiles is defined to use a minor-only version numbering convention. That is, for a given profile name, each successive version MUST be compatible with all earlier versions. Any incompatible change to a profile MUST use a different profile name.

For one version of a profile to be considered compatible with another version, the later version MUST be a strict superset of the earlier version. This requires the following of the later version with respect to all earlier versions to which it is to be compatible:

- The later version MAY add requirements that were not in earlier versions of the profile, but MUST NOT remove requirements.
- The later version MAY remove one or more conditions that had previously been placed on a requirement. For example, if a previous profile REQUIRED X only if condition A was True, then the later profile might require X unconditionally.

For profiles, the following notation is defined to identify specific versions:

Notation	Description	Example
ProfileName:Version	Refers to a specific version of the profile.	Baseline:1
ProfileName	Refers to any version of the profile.	Baseline

ProfileName MUST start with a letter or underscore, and subsequent characters MUST be letters, digits, underscores or hyphens. The terms "letter" and "digit" are as defined in Appendix B of the XML specification [8].

2.3.4 Baseline Profiles

For every Service Object (and Root Object) there SHOULD be at least one profile defined. In many cases it is desirable to define a Baseline profile that indicates the minimum requirements REQUIRED for any device that supports that object. Where a Baseline profile is defined, and if the (DEPRECATED) DeviceSummary parameter (Section 3.7) is fully implemented, it would normally be expected that all implementations of the corresponding object would indicate support for the Baseline profile in addition to any other profiles supported.

2.3.5 Types of Requirements in a Profile

Because a profile is defined within the context of a single object (and major version), all of the requirements associated with the profile MUST be specific to the data model associated with that object.

Profile requirements can include any of the following types of requirements associated with an object's data model:

- A requirement for read support of a Parameter.
- A requirement for write support of a Parameter.
- A requirement for support of a sub-object contained within the overall object.
- A requirement for the ability to add or remove instances of a sub-object.
- A requirement to support active notification for a Parameter.
- A requirement to support access control for a given Parameter.

For each of the requirement categories listed above, a profile can define the requirement unconditionally, or can place one or more conditions on the requirement. For example, a profile might require that a Parameter be supported for reading only if the device supports some other parameter or object (one that is not itself REQUIRED by the profile). Such conditions will be directly related to the data model of the overall object associated with the profile.

Because a device has to be able to support multiple profiles, all profiles MUST be defined such that they are non-contradictory. As a result, profiles MUST only define minimum requirements to be met, and MUST NOT specify negative requirements. That is, profiles will not include requirements that specify something that is not to be supported by the device, or requirements that exclude a range of values.

2.4 DEPRECATED and OBSOLETED Items

The key word "DEPRECATED" in the data model definition for any TR-069-enabled [2] device is to be interpreted as follows: This term refers to an object, parameter or parameter value that is defined in the current version of the standard but is meaningless, inappropriate, or otherwise unnecessary. It is intended that such objects, parameters or parameter values will be removed from the next major version of the data model. Requirements on how to interpret or implement deprecated objects, parameters or parameter values are given below. For more information on how to interpret or implement specific deprecated objects, parameters or parameter values, refer to the definition of the object or parameter.

The key word "OBSOLETED" in the data model definition for any TR-069-enabled [2] device is to be interpreted as follows: This term refers to an object, parameter or parameter value that meets the requirements for being deprecated, and in addition is obsolete. Such objects, parameters or parameter values can be removed from a later minor version of a data model, or from a later version of a profile, without this being regarded as breaking backwards compatibility rules. Requirements on how to interpret or implement obsoleted objects, parameters or parameter values are given below. For more information on how to interpret or implement specific obsoleted objects, parameters or parameter values, refer to the definition of the object or parameter.

2.4.1 Requirements for DEPRECATED Items

This section defines requirements that apply to all DEPRECATED objects, parameters and parameter values unless specifically overridden by the object or parameter definition.

Data model requirements:

1) The definition of a DEPRECATED parameter, object or parameter value MUST include an explanation of why the item is deprecated.
2) The definition of a DEPRECATED parameter, object or parameter value MAY specify further requirements relating to the item; such requirements MAY override CPE or ACS requirements specified in this section.

CPE requirements:

1) A DEPRECATED parameter MUST have a value which is valid for its data type and fulfils any range (for numeric parameters), length (for string, base64 or hexBinary parameters) and enumerated value (for string parameters) requirements.
2) Detailed behavioral requirements for a DEPRECATED parameter, e.g. that its value is a unique key, MAY be ignored by the CPE.
3) The CPE MUST, if such operations are permitted by the data model definition, permit creation of DEPRECATED objects, modification of DEPRECATED parameters, and setting of DEPRECATED parameter values. However, it MAY choose not to apply such changes to its operational state.
4) Regardless of whether DEPRECATED changes are applied to the CPE operational state, a read of a DEPRECATED writable parameter SHOULD return the value that was last written, i.e. the CPE is expected to store the value even if it chooses not to apply it to its operational state.
5) The CPE MAY reject an attempt by the ACS to set any parameter to a DEPRECATED value.

ACS requirements:

1) The ACS SHOULD NOT create DEPRECATED objects, modify DEPRECATED parameters, or set DEPRECATED parameter values.
2) The ACS SHOULD ignore DEPRECATED objects, parameters and parameter values.
3) The ACS SHOULD NOT set a DEPRECATED parameter to a value that is invalid for its data type or fails to fulfill any range (for numeric parameters), length (for string, base64 or hexBinary parameters) or enumerated value (for string parameters) requirements.
4) The ACS SHOULD NOT set any parameter to a DEPRECATED value.

2.4.2 Requirements for OBSOLETED Items

This section defines requirements that apply to all OBSOLETED objects, parameters or parameter values unless specifically overridden by the object or parameter definition.

An OBSOLETED object, parameter or parameter value MUST meet all the requirements of the previous section. In addition, the following data model requirements apply.

1) An OBSOLETED object, parameter or parameter value MAY be removed from a later minor version of a data model without this being regarded as breaking backwards compatibility rules.
2) An OBSOLETED object, parameter or parameter value MUST NOT be removed from the current version of a profile, but MAY be removed from a later version of a profile without this being regarded as breaking backwards compatibility rules.
3) A data model definition MUST include a list of those OBSOLETED objects, parameters or parameter values that have been removed from the data model or from its profiles. This is to prevent future namespace conflicts.

3 Object Definitions

3.1 General Notation

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.

Parameter names MUST be treated as case sensitive. The name of each node in the hierarchy MUST start with a letter or underscore, and subsequent characters MUST be letters, digits, underscores or hyphens. The terms "letter" and "digit" are as defined in Appendix B of the XML specification [8].

Where multiple instances of an object can occur, the placeholder node name {i} is shown. In actual use, this placeholder is to be replaced by an Instance Identifier.

3.2 Data Types and Representation

Parameters make use of a limited subset of the default SOAP data types [5]. The supported data types are defined by the DM Schema and are also listed in A.2.3.

The named data types that specify the representations of IP addresses, MAC addresses etc, are defined in a DM Instance document (see Annex A). Table 1 gives links to this XML file and to the corresponding HTML. Note that this DM Instance defines named data types that are expected to be used in several data model definitions; it is possible to define local named data types in any DM Instance document.

Table 1 – Named Data Type Definitions

DM Instance	XML and HTML
tr-106-1-0-types.xml	http://broadband-forum.org/cwmp/tr-106-1-0-types.xml
	http://broadband-forum.org/cwmp/tr-106-1-0-types.html

The following sub-sections specify additional rules governing parameter value representation within XML documents.

3.2.1 Escaping non-ASCII Characters

When transporting a string value within an XML document, any characters which are special to XML MUST be escaped as specified by the XML specification [8]. Additionally, any characters other than printable ASCII characters, i.e. any characters whose decimal ASCII representations are outside the (inclusive) ranges 9-10 and 32-126, SHOULD be escaped as specified by the XML specification.

3.2.2 Date and Time Rules

All times MUST be expressed in UTC (Universal Coordinated Time) unless explicitly stated otherwise in the definition of a parameter of this type.

If absolute time is not available to the CPE, it SHOULD instead indicate the relative time since boot, where the boot time is assumed to be the beginning of the first day of January of year 1, or 0001 01 01T00:00:00. For example, 2 days, 3 hours, 4 minutes and 5 seconds since boot would be expressed as 0001 01 03T03:04:05. Relative time since boot MUST be expressed using an untimezoned representation. Any untimezoned value with a year value less than 1000 MUST be interpreted as a relative time since boot.

If the time is unknown or not applicable, the following value representing "Unknown Time" MUST be used: 0001-01-01T00:00:00Z. For an infinite timeline, the following value representing "Infinite Time" MUST be used: 9999-12-31T23:59:59Z.

Any dateTime value other than one expressing relative time since boot (as described above) MUST use timezoned representation (that is, it MUST include a timezone suffix).

3.2.3 Comma-separated Lists

For strings that are defined to contain comma-separated lists, the format is defined as follows. Between every pair of successive items in a comma-separated list there MUST be a separator. The separator MUST include exactly one comma character, and MAY also include one or more whitespace characters before or after the comma. The entire separator, including any whitespace characters, MUST NOT be considered part of the list items it separates. The last item in a comma-separated list MUST NOT be followed with a separator. Individual items in a comma-separated list MUST NOT include a whitespace or comma character within them. Any whitespace or comma characters within an item value MUST be escaped using percent encoding as specified in Section 2.1/RFC 3986 [7].

3.2.4 Parameters that Reference Parameters or Objects

For string parameters that are defined to contain the hierarchical Path Name of an object (or for each item in parameters that are defined to contain comma-separated lists of object Path Names), the representation of the object name MUST NOT include a trailing "dot." An example of a parameter of this kind in the TR-098 [3] InternetGatewayDevice:1 data model is InternetGatewayDevice.Layer3Forwarding.DefaultConnectionService. For this parameter, the following is an example of a properly formed value:

InternetGatewayDevice.WANDevice.1.WANConnectionDevice.2.WANPPPConnection.1

Path Names in parameter values MUST always be full Path Names, with the single exception that a path that begins with a dot is relative to the Root or Service Object. For example, in the Device Root Object, a parameter value of ".DeviceInfo" always means "Device.DeviceInfo".

In order to be able to use reference parameters as unique keys (A.2.8.1), their Path Names MUST conceptually be converted to full Path Names before being compared. For example, in the Device Root Object, ".DeviceInfo." and "Device.DeviceInfo." would compare as equal. If a reference parameter is list-valued, i.e. it is a list of Path Names or Instance Numbers, the parameter value MUST conceptually be regarded as a set when being compared, i.e. the comparison has to ignore the item order and any repeated items. For example, "1,2,1" and "2,1" would compare as equal because both reference instances 1 and 2.

A CPE that supports the Alias-Based Addressing Mechanism (defined in Section 3.6.1/TR-069 and described in Appendix II/TR-069 [2]), MUST support the ability to use Instance Alias object addressing in reference values. For example, the following paths illustrate a reference to the same object instance using an Instance Number and then an Instance Alias:

Object.SomeReferenceParameter = "Object.FooObject.5"

Object.SomeReferenceParameter = "Object.FooObject.[Text-A]"

In the first example, the reference points to the FooObject with Instance Number 5. In the second example, the reference points to the FooObject with an Instance Alias of "Text-A".

References are defined as strong or weak in the data model. A strong reference always either references an existing parameter or object, or else is a null reference. On the other hand, a weak reference does not necessarily reference an existing parameter or object.

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

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

The following requirements apply when a reference parameter contains Instance Aliases (as defined in Section A.2.2/TR-069 [2]):

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

3.2.5 Units Conventions

For numeric parameters whose values are defined in terms of units, bit and byte-related units will always refer to powers of 2. For example, a kilobyte will always be 1024 bytes, a megabyte always 1024 * 1024 bytes, etc.

3.2.6 Default Maximum String Length

For string-valued parameters, 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. Similarly, for strings in which the content is a pattern, the longest possible matching value determines the maximum length. For strings in which the content is a list, the maximum number of items and the individual item lengths can help to determine the maximum string length. If a string does not have an explicit or implied maximum length, the default maximum is 16 characters unless otherwise indicated in an associated parameter description.

3.3 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 not contained within another vendor-specific 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. The OUI or domain name used for a given vendor-specific parameter MUST be one that is assigned to the organization that defined this parameter (which is not necessarily the same as the vendor of the CPE or ACS). An OUI is an

organizationally unique identifier as defined in [4], which MUST be 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 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_".

The name of a vendor-specific parameter or object that is contained within another vendor-specific object which itself begins with the prefix described above need not itself include the prefix.

The full Path Name of a vendor-specific parameter or object MUST NOT exceed 256 characters in length.

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

Device.UserInterface.X_012345_AdBanner
Device.X_EXAMPLE-COM_MyConfig.Status

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.

3.4 Object Definitions (Removed)

Object Definitions moved to TR-181 [13].

3.5 Inform Requirements (Removed)

Inform Requirements moved to TR-181 [13].

3.6 Notification Requirements (Removed)

Notification Requirements moved to TR-181 [13].

3.7 DeviceSummary Definition

Note – the DeviceSummary parameter is DEPRECATED. This is because the TR-157 [12] SupportedDataModel object (see Section 2.1.3) and associated Device Type XML documents (DT Instances) provide a more granular and scalable way of describing the device's data model than does DeviceSummary. Therefore the value of DeviceSummary MAY be an empty string if (and only if) the SupportedDataModel object is supported[4].

The DeviceSummary parameter is defined to provide an explicit summary of the top-level data model of the device, including version and profile information. This parameter MAY be used by an ACS to discover the nature of the device and the ACS's compatibility with specific objects supported by the device.

The DeviceSummary is defined as a list that includes the Root Object followed by all Service Object instances (or support for a Service Object type if no instances are currently present). For each of these objects, the DeviceSummary specifies the version of the object, the associated Instance Number used to identify the specific object instance, and a list of the supported profiles for that object.

The syntax of the DeviceSummary parameter is defined formally as follows:

[4] The (DEPRECATED) DeviceSummary parameter applies only to major version 1 of the "Device" Root Object.


```
DeviceSummary = RootObject [", " ServiceObject]*

RootObject = ObjectName ":" ObjectVersion "[](" ProfileList ")"

ServiceObject = ObjectName ":" ObjectVersion "[" [Instance] "](" ProfileList ")"

ObjectVersion = MajorVersion "." MinorVersion

ProfileList = [Profile [ ", " Profile]*]

Profile = ProfileName ":" ProfileVersion

MajorVersion = Integer

MinorVersion = Integer

ProfileVersion = Integer

Integer = DIGIT*

Instance = [ "+" ] NONZERODIGIT [DIGIT]*
```

For each object instance, the ObjectVersion element MUST indicate the major and minor versions of the object supported by the device.

The ObjectVersion for all objects for which explicit major and minor version numbers have not been defined is 1.0. Future updates to these objects will specify distinct version numbers.

Instance is the Instance Number of the particular object instance. If the device supports an object type, but no instances are currently present, a single entry for this object MUST be listed in the DeviceSummary, and the Instance Number MUST be empty ("[]"). In this case, the device need not list support for specific profiles since the profile list might be dependent on the specific instance when it is instantiated.

If the Instance Number for an object might change (for example, if the instances represent physically separate devices, being managed by proxy, that can be connected or disconnected), the Instance Number MUST be prefixed with a "+" character. Lack of a "+" character indicates that the Instance Number is expected to remain unchanged.

For each object (Root Object and Service Objects), a device MUST list all profiles that it supports in the ProfileList element. That is, it MUST list all profiles for which the device's actual level of support is a superset. Each entry in the ProfileList MUST include the ProfileName and the ProfileVersion. The ProfileVersion is a single integer representing the minor version of the profile.

Vendor-specific objects and profiles MAY be included in this list, and if so MUST begin with X_<VENDOR>_, where <VENDOR> MUST be as defined in Section 3.3.

3.7.1 DeviceSummary Examples

Below are some examples of the DeviceSummary parameter. (The first examples correspond directly to the examples given in Section 2.1.2.)

Simple device supporting the ABCService Service Object:

"Device:1.0[](Baseline:1), ABCService:1.0[1](Baseline:1)"

Device supporting both ABCService and XYZService Service Objects:

"Device:1.0[](Baseline:1), ABCService:1.0[1](Baseline:1), XYZService:1.0[1](Baseline:1)"

TR-098 [3] Internet Gateway Device that also supports the ABCService and XYZService Service Objects:

"InternetGatewayDevice:1.0[](Baseline:1), ABCService:1.0[1](Baseline:1), XYZService:1.0[1](Baseline:1)"

Device supporting the ABCService Service Object and proxying for two devices supporting the functionality of the XYZService Service Object:

"Device:1.0[](Baseline:1), ABCService:2.17[1](Baseline:1), XYZService:1.2[1](Baseline:2), XYZService:1.2[2](Baseline:2, AnotherProfile:3)"

TR-098 [3] Internet Gateway Device also serving as a management proxy for three devices supporting the functionality of the ABCService Service Object:

"InternetGatewayDevice:1.0[](Baseline:1), ABCService:1.0[1](Baseline:1), ABCService:1.0[2](Baseline:1), ABCService:1.0[3](Baseline:1, AnotherProfile:1)"

TR-098 [3] Internet Gateway Device with no additional service objects supported:

"InternetGatewayDevice:1.0[](Baseline:1)"

Device supporting the ability to proxy for devices supporting the functionality of the ABCService Service Object, but with no current instances of that object:

"Device:1.0[](Baseline:1), ABCService:2.17[]()"

Device supporting the ABCService Service Object with the baseline and a vendor-specific profile:

"Device:1.0[](Baseline:1), ABCService:2.17[1](Baseline:1, X_EXAMPLE-COM_MyProfile:2)"

Device supporting the ABCService Service Object, but with no profiles:

"Device:1.0[](Baseline:1), ABCService:2.17[1]()"

3.8 Alias Parameter Requirements

This section defines requirements that apply to Alias Parameters (Section A.2.2.2/TR-069 [2]); how they are defined and when to include them within an Object definition.

3.8.1 Alias Parameter Definition

The Alias Parameter value MUST be a string as defined in Section 3.2.1/XML Schema Part 2 [14]. Specifically, it has a length from 1 to 64 characters (so it cannot be empty); the supported character set is defined as numbers (ASCII 0x30-0x39), letters (ASCII 0x41-0x5A and ASCII 0x61-0x7A), underscore (ASCII 0x5F) and dash (ASCII 0x2D); it is case sensitive; the first character MUST be a letter (ASCII 0x41-0x5A or ASCII 0x61-0x7A).

The Alias Parameter MUST be a non-functional unique key for the table (see A.2.8.1).

3.8.2 Support of the Alias Parameter

Although an Alias Parameter has special functions used by the Alias-Based Addressing Mechanism (Section 3.6.1/TR-069 [2]), it MUST also behave as a generic TR-069 parameter. In particular:

- All the XML data models MUST define Alias Parameters for all their Multi-Instance Objects that are required to support it (see Section 3.8.3).
- The Alias Parameter MUST always be named "Alias".

For example, the following parameter path can be used to get/set the Alias Parameter of the Multi-Instance Object:

Device.Bridging.Bridge.1.Alias

3.8.3 Multi-Instance Objects Alias Parameter Requirements

The presence of an Alias Parameter is specifically REQUIRED on every Multi-Instance Object that falls under the following definition:

- All writable Multi-Instance Objects (i.e. objects that can be created and/or deleted by the ACS).

- All Multi-Instance Objects which have writable parameters.

The following Multi-Instance Objects MAY have an Alias Parameter:

- All Multi-Instance Objects that are built-in objects. A built-in object is an object which exists in the CPE's factory configuration. Built-in objects do not appear or disappear without a firmware upgrade or hardware change. Built-in objects MAY have Alias Parameters that MUST follow the unique key requirements defined in A.2.8.1.

The following Multi-Instance Objects MUST NOT have an Alias Parameter:

- Transient objects. An object is considered transient if it represents read-only state parameters which are typically accessed by the ACS by retrieving all instances at once (e.g. diagnostic results). Transient objects can also represent status which is very likely to change between two successive CWMP sessions (such as TR-104 Sessions).

4 Profile Definitions (Removed)

Profile Definitions moved to TR-181 [13].

5 Normative References

A list of the currently valid Broadband Forum Technical Reports is published at http://www.broadband-forum.org. The following documents are referenced by this specification.

[1] RFC 2119, *Key words for use in RFCs to Indicate Requirement Levels*, IETF, 1997

[2] TR-069 Amendment 4, *CPE WAN Management Protocol*, Broadband Forum, 2011

[3] TR-098 Amendment 2, *Internet Gateway Device Data Model for TR-069*, Broadband Forum, 2008

[4] *Organizationally Unique Identifiers (OUIs)*, IEEE

[5] *Simple Object Access Protocol (SOAP) 1.1*, W3C, 2000

[6] RFC 3513, *Internet Protocol Version 6 (IPv6) Addressing Architecture*, IETF, 2003

[7] RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*, IETF, 2005

[8] *Extensible Markup Language (XML) 1.0 (Fourth Edition)*, W3C, 2008

[9] RFC 2648, *A URN Namespace for IETF Documents*, IETF, 1999

[10] TR-143 Corrigendum 1, *Enabling Network Throughput Performance Tests and Statistical Monitoring*, Broadband Forum, 2008

[11] *XML Schema Part 0: Primer Second Edition*, W3C, 2004

[12] TR-157 Amendment 3, *Component Objects for CWMP*, Broadband Forum, 2010

[13] TR-181 Issue 1 and TR-181 Issue 2 Amendment 3, *Device Data Model for TR-069*, Broadband Forum

[14] *XML Schema Part 2: Datatypes Second Edition*, W3C, 2004

[15] TR-140 Issue 1.1 TR-069 Data Model for Storage Service Enabled Devices, Broadband Forum, 2007

Annex A. CWMP Data Model Definition XML Schema

A.1 Introduction

The CWMP Data Model Definition XML Schema [11], or DM Schema, is used for defining TR-069 [2] data models, and is specified in A.3.

DM Schema instance documents can contain any or all of the following:

- Data type definitions
- Root Object definitions (including profiles)
- Service Object definitions (including profiles)
- Component definitions
- Vendor extension definitions

A.2 Normative Information

It is possible to create instance documents that conform to the DM Schema but nevertheless are not valid data model definitions. This is because it is not possible to specify all the normative data model definition requirements using the XML Schema language. Therefore, the schema contains additional requirements written using the usual normative language. Instance documents that conform to the DM Schema and meet these additional requirements are referred to as DM Instances.

For example, the definition of the parameter element includes the following additional requirements on the name and base attributes:

```
  <xs:complexType name="ModelParameter">
    <xs:annotation>
      <xs:documentation>Parameter definition and reference.</xs:documentation>
    </xs:annotation>
    …
    <xs:attribute name="name" type="tns:ParameterName">
      <xs:annotation>
        <xs:documentation>MUST be unique within the parent object (this is checked by schema
                          validation).
MUST be present if and only if defining a new parameter.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="base" type="tns:ParameterName">
      <xs:annotation>
        <xs:documentation>MUST be present if and only if modifying an existing
                          parameter.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    …
  </xs:complexType>
```

In some cases, a requirement that is in fact implied by the DM Schema is emphasized within the schema via the xs:documentation element (the uniqueness requirement on the name is an example of this).

In other cases, a schema-implied requirement is not highlighted. For example, the name and base attributes are of type tns:ParameterName:

```
<!DOCTYPE cwmp-datamodel [
  …
  <!ENTITY name "([\i-[:]][\c-[:\.]]*)">
  …
]>
  …
  <xs:simpleType name="ParameterName">
    <xs:annotation>
      <xs:documentation>Parameter name (maximum length 256); the same as xs:NCName except that periods
                        are not permitted.  This name MUST in addition follow the vendor-specific
                        parameter name requirements of Section 3.3.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:token">
      <xs:maxLength value="256"/>
      <xs:pattern value="&name;"/>
    </xs:restriction>
  </xs:simpleType>
```

This states that the parameter name is a string that follows the following rules:

- It is derived from xs:token, which has a whitespace facet of "collapse", meaning that any leading whitespace in the name will be ignored.
- It has a maximum length of 256 characters.
- Its first character matches the pattern "[\i-[:]]", which means "any character permitted as the first character of an XML name, except for a colon", and any subsequent characters match the pattern "[\c-[:\.]]", which means "any character permitted in an XML name, except for a colon and a dot".
- It follows the vendor-specific parameter name requirements of Section 3.3.

The question of the location of the definitive normative information therefore arises. The answer is as follows:

- All the normative information in the main part of the document remains normative.
- The DM Schema, and the additional requirements therein, are normative. Some of these additional requirements are duplicated (for emphasis) in this Annex.
- The DM Schema references additional material in this Annex. Such material is normative.
- If the DM Schema conflicts with a normative requirement in the main part of the document, this is an error in the DM Schema, and the requirement in the main part of the document takes precedence.

A.2.1 Importing DM Instances

DM Instances are imported using the top-level import element. The DM Schema specifies that the DM Instance is located via the file attribute.

The rules governing the file attribute's value and its use for locating the DM Instance are as follows:

- It MUST be a URL adhering to RFC 3986 [7].
- If the URL includes a scheme, it MUST be http, https or ftp.
- If the URL includes an authority, it MUST NOT include credentials.
- For standard BBF DM Instances, the rules that apply to the filename part (final path segment) of the A.2.1.1 BBFURL MUST be applied to the filename part of this URL. This means that the corrigendum number can be omitted in order to refer to the latest corrigendum.
- If the URL is a relative reference, processing tools MUST apply their own logic, e.g. apply a search path.

A.2.1.1 URI Conventions

The top-level spec attribute contains the URI of the associated specification document, e.g. the BBF Technical Report.

This URI SHOULD uniquely identify the specification. More than one DM Schema instance document MAY reference the same specification.

The top-level file attribute contains the name of the DM Schema instance document, e.g. the XML file that defines a given version of a data model.

The following rules apply to the value of the top-level spec attribute:

- For a BBF Technical Report, it MUST be of the form "urn:broadband-forum-org:tr-nnn-i-a-c", where nnn is the specification number (including leading zeros), i is the issue number, a is the amendment number, and c is the corrigendum number. The issue, amendment and corrigendum numbers do not include leading zeros. For example, "urn:broadband-forum-org:tr-106-1-0" refers to TR-106 (Issue 1 Amendment 0), and "urn:broadband-forum-org:tr-106-1-2" refers to TR-106 (Issue 1) Amendment 2.
- For specifications issued by other standards organizations, or by vendors, it SHOULD be of a standard form if one is defined. For example, RFC 2648 [9] specifies a syntax for referencing RFCs.
- Note that processing tools are likely to assume that all files that share a spec value are related to each other. Therefore, use of meaningful spec values is RECOMMENDED.

The following rules apply to the value of the top-level file attribute.

- For a BBF Technical Report, it MUST be of the form "tr-nnn-i-a-c.xml" or "tr-nnn-i-a-c-label.xml", where nnn, i, a and c are the same as in the spec attribute. The label, which MUST NOT begin with a digit, is not needed if only one DM Schema instance document is associated with the specification.
- It SHOULD be the same as the actual file name (omitting the directory name). Under some circumstances this will not be possible, e.g. because the content is stored in a database and not in a file system.

Formally, the values of the spec and file attributes are defined as follows:


```
SpecAttr = SpecURI

FileAttr = FileName

SpecURI = BBFURI
        | OtherURI

BBFURI = "urn:broadband-forum-org:" BBFDoc

FileName = BBFDoc BBFSubDoc ".xml"

BBFDoc = "tr-" BBFNumber BBFIssue BBFAmendment BBFCorrigendum

BBFNumber = [DIGIT]{3,} // including leading zeros, e.g. 069

BBFIssue = "-" NoLeadingZeroPositiveNumber

BBFAmendment = "-" NoLeadingZeroNumber

BBFCorrigendum = "-" NoLeadingZeroNumber

BBFSubDoc = "-" LABEL   // distinguishing label (not beginning with a digit)
          | ""          // not needed if only one DM Instance is associated with spec

NoLeadingZeroNumber = [DIGIT]
                    | [NONZERODIGIT] [DIGIT]*

NoLeadingZeroPositiveNumber = [NONZERODIGIT] [DIGIT]*

OtherURI = <of a standard form if one is defined>
```

Standard BBF DM Instances can be accessed at the following URL:

```
BBFURL = "http://www.broadband-forum.org/cwmp/" FileName

FileName = <as before, except that BBFCorrigendum is modified as follows:>

BBFCorrigendum = "-" NoLeadingZeroNumber
               | ""     // if omitted, most recent corrigendum is assumed
```

For example, the DM Instance associated with TR-106 Amendment 2 can be accessed at http://www.broadband-forum.org/cwmp/tr-106-1-2.xml. If two DM Instances had been associated with TR-106 Amendment 2, they might have been accessible at http://www.broadband-forum.org/cwmp/tr-106-1-2-types.xml and http://www.broadband-forum.org/cwmp/tr-106-1-2-objects.xml.

A.2.2 Descriptions

Many elements have descriptions, and the same rules apply to all description elements in the DM Schema. A description is free text which can contain a limited amount of MediaWiki-like markup as specified in A.2.2.3.

A.2.2.1 Character Set

For BBF standards, the character set MUST be restricted to printable characters in the Basic Latin Unicode block, i.e. to characters whose decimal ASCII representations are in the (inclusive) ranges 9-10 and 32-126.

A.2.2.2 Pre-processing

All DM Instance processing tools MUST conceptually perform the following pre-processing before interpreting the markup:

1) Remove any leading whitespace up to and including the first line break[5].

[5] It can be assumed that all line breaks are represented by a single line feed, i.e. ASCII 10. See [8] Section 2.11.

2) Remove the longest common whitespace prefix (i.e. that occurs at the start of every line) from each line. See the example below, where three lines start with four spaces and one line starts with five spaces, so the longest whitespace prefix that occurs at start of each line is four spaces. In this calculation, a tab character counts as a single character. To avoid confusion, the description SHOULD NOT contain tab characters.

3) Remove all trailing whitespace, including line breaks.

This pre-processing is designed to permit a reasonable variety of layout styles while still retaining predictable behavior. For example, both the following:

```
<description>This is the first line.
This is the second line.
 This is the indented third line.
This is the fourth line.</description>
```

```
<description>
    This is the first line.
    This is the second line.
     This is the indented third line.
    This is the fourth line.
</description>
```

…result in the following:

```
This is the first line.
This is the second line.
 This is the indented third line.
This is the fourth line.
```

A.2.2.3 Markup

The pre-processed description can contain the following markup, which is inspired by, but is not identical to, MediaWiki markup. All DM Instance processing tools SHOULD support this markup to the best of their ability.

Table 2 – XML Description Markup

Name	Markup Example	Description
Italics	''italic text''	Two apostrophes on each side of some text will result in the contained text being emphasized in italics.
Bold	'''bold text'''	Three apostrophes on each side of some text will result in the contained text being emphasized in bold.
Bold italics	'''''b+i text'''''	Five apostrophes on each side of some text will result in the contained text being emphasized in bold italics.
Paragraph	This paragraph just ended.	A line break is interpreted as a paragraph break.
Bulleted lists	* level one ** level two * level one again ** level two again *** level three *: level one continued outside of list	A line starting with one or more asterisks (*) denotes a bulleted list entry, whose indent depth is proportional to the number of asterisks specified. If the asterisks are followed by a colon (:), the previous item at that level is continued, as shown. An empty line, or a line that starts with a character other than an asterisk, indicates the end of the list.
Numbered lists	# level one ## level two # level one again ## level two again ### level three #: level one continued outside of list	A line starting with one or more number signs (#) denotes a numbered list entry. All other conventions defined for bulleted lists apply here (using # rather than *), except that numbered list entries are prefixed with an integer decoration rather than a bullet.

Name	Markup Example	Description

Indented lists	`: level one`
	`:: level two`
	`: level one again`
	`:: level two again`
	`::: level three`
	`outside of list`
		A line starting with one or more colons (:) denotes an indented list entry.
		All other conventions defined for bulleted lists apply here (using : rather than *), except that indented list entries have no prefix decoration, and item continuation is not needed.

Verbatim	`code example:`
	` if (something) {`
	`  /* do something */`
	` } else {`
	`  /* do other */`
	` }`
		A block of lines each of which starts with a space is to be formatted exactly as typed, preferably in a fixed width font.
		This allows code fragments, simple tables etc. to be included in descriptions.
		Note that the pre-processing rules of A.2.2.2 imply that it is not possible to process an entire description as verbatim text (because all the leading whitespace would be removed). This is not expected to be a problem in practice.

Hyperlinks	`http://www.broadband-forum.org`	URL links are specified as plain old text (no special markup).

Templates	`{{bibref|1|section 2}}`
	`{{section|table}}`
	`{{param|Enable}}`
	`{{enum|Error}}`
		Text enclosed in double curly braces ({}) is a template reference, which is replaced by template-dependent text.
		A.2.2.4 specifies the standard templates.

A.2.2.4 Templates

A template invocation is encoded as two curly braces on either side of the template name and arguments. Arguments can follow the template name, separated by vertical pipe (|) characters. All whitespace is significant. For example:

```
{{someTemplate|arg1|arg2|…|argN}}
```

In some cases, one template can impact the behavior of another template, e.g. the definitions of both the `{{enum}}` and the `{{hidden}}` templates state that the template expansion can be automatically placed after the rest of the description, which raises the question of which template expansion would come first. This ambiguity is resolved by stating that processing tools SHOULD generate such automatic text in the same order that the templates are defined below. In the above example, `{{enum}}` is defined before `{{hidden}}`, so an automatically-generated list of enumeration values would be placed before an automatically-generated explanation that the parameter value is hidden.

The following standard templates are defined. Any vendor-specific template names MUST obey the rules of Section 3.3.

Table 3 – XML Description Templates

Name	Markup Definition	Description

Bibliographic reference	`{{bibref|id}}`
	`{{bibref|id|section}}`
		A bibliographic reference.
		The id argument MUST match the id attribute of one of the current file's (or an imported file's) top-level bibliography element's reference elements (A.2.4).
		The OPTIONAL section argument specifies the section number, including any leading "section", "annex" or "appendix" text.
		Typically, processing tools will (a) validate the id, and (b) replace the template reference with something like "[id] section".
		Markup examples:
		`{{bibref|1}}`
		`{{bibref|1|section 3}}`

Name	Markup Definition	Description
Section separator	{{section|category}} {{section}}	The beginning or end of a section or category. This is a way of splitting the description into sections. If the category argument is present, this marks the end of the previous section (if any), and the beginning of a section of the specified category. The "table", "row" and "examples" categories are reserved for the obvious purposes. If the category argument is absent, this marks the end of the previous section (if any). Typically, processing tools will (a) validate the category, and (b) replace the template reference with a section marker. Markup examples: {{section|table}} {{section|row}} {{section|examples}}
Number of entries parameter description	{{numentries}}	A description of a "NumberOfEntries" parameter. This template SHOULD be used for all such parameters. It will be expanded to something like "The number of entries in the <table> table.". In most cases, the description will consist only of {{numentries}} but it MAY be followed by additional text if desired.
Parameter and object reference	{{param|ref}} {{param|ref|scope}} {{param}} {{object|ref}} {{object|ref|scope}} {{object}}	A reference to the specified parameter or object. The OPTIONAL ref and scope arguments reference a parameter or object. Scope defaults to normal. Parameter and object names SHOULD adhere to the rules of A.2.3.4. Typically, processing tools will (a) validate the reference, and (b) replace the template reference with the ref argument or, if it is omitted, the current parameter or object name, possibly rendered in a distinctive font. Processing tools can use the scope to convert a relative path into an absolute path in order, for example, to generate a hyperlink. Markup examples: {{param|Enable}} {{object|Stats.}}
Profile reference	{{profile|ref}} {{profile}}	A reference to the specified profile. The OPTIONAL ref argument references a profile. Typically, processing tools will (a) validate the reference, and (b) replace the template reference with the ref argument or, if it is omitted, the current profile name, possibly rendered in a distinctive font. Markup examples: {{profile|Baseline:1}} {{profile}}
List description	{{list}} {{list|arg}} {{nolist}}	A description of the current parameter's list attributes. This template SHOULD only be used within the description of a list-valued parameter (A.2.7.1). This is a hint to processing tools to replace the template reference with a description of the parameter's list attributes. This overrides processing tools' expected default behavior (unless suppressed by {{nolist}}) of describing the list attributes before the rest of the description. The OPTIONAL argument specifies a fragment of text that describes the list and SHOULD be incorporated into the template expansion. Typically processing tools will generate text of the form "Comma-separated list of <dataType>." Or "Comma-separated list of <dataType>, <arg>.".

Name	Markup Definition	Description
Reference description	{{reference}} {{reference|arg}} {{noreference}}	A description of the object or parameter that is referenced by the current parameter. This template SHOULD only be used within the description of a reference parameter (A.2.3.7). This is a hint to processing tools to replace the template reference with a description of the parameter's reference attributes. This overrides processing tools' expected default behavior (unless suppressed by {{noreference}}) of describing the reference attributes after the list attributes (for a list-valued parameter) or before the rest of the description (otherwise). The OPTIONAL argument is relevant only for a pathRef; it specifies a fragment of text that describes the referenced item and SHOULD be incorporated into the template expansion. Typically processing tools will generate text of the form "The value MUST be the full path name of <arg>…", in which the generated text can be expected to be sensitive to whether or not the parameter is list-valued.
Named data type	{{datatype}} {{datatype|arg}} {{nodatatype}}	A description of the current parameter's named data type. This template SHOULD only be used within the description of a parameter of a named data type (A.2.3.1). This is a hint to processing tools to replace the template reference with an indication of the parameter's named data type, possibly including additional details or a hyperlink to such details. This overrides processing tools' expected default behavior (unless suppressed by {{nodatatype}}) of describing the named data type before the rest of the description. The OPTIONAL argument affects how the data type is described. If it has the literal value "expand", processing tools SHOULD replace the template reference with the actual description of the named data type (as opposed to referencing the description of the named data type).
Profile description	{{profdesc}} {{noprofdesc}}	An auto-generated description of a profile. This template SHOULD only be used within the description of a profile (A.2.9). This is a hint to processing tools to replace the template reference with a description of the profile. This overrides processing tools' expected default behavior (unless suppressed by {{noprofdesc}}) of describing the profile before the rest of the description. Typically processing tools will generate text of the form "This table defines the <profile:v> profile for the <object:m> object. The minimum REQUIRED version for this profile is <object:m.n>." (or more complex text if the profile is based on or extends other profiles).

Name	Markup Definition	Description
Enumeration reference	{{enum|value}} {{enum|value|param}} {{enum|value|param|scope}} {{enum}} {{noenum}}	A reference to the specified enumeration value. The OPTIONAL value argument specifies one of the enumeration values for the referenced parameter. If present, it MUST be a valid enumeration value for that parameter. The OPTIONAL param and scope arguments identify the referenced parameter. Scope defaults to normal. If present, param SHOULD adhere to the rules of A.2.3.4. If omitted, the current parameter is assumed. If the arguments are omitted, this is a hint to processing tools to replace the template reference with a list of the parameter's enumerations, possibly preceded by text such as "Enumeration of:". This overrides processing tools' expected default behavior (unless suppressed by {{noenum}}) of listing the parameter's enumerations after the rest of the description. Otherwise, typically processing tools will (a) validate that the enumeration value is valid, and (b) replace the template reference with the value and/or param arguments, appropriately formatted and with the value possibly rendered in a distinctive font. Processing tools can use the scope to convert a relative path into an absolute path in order, for example, to generate a hyperlink. Markup examples: {{enum|None}} {{enum|None|OtherParam}}
Pattern reference	{{pattern|value}} {{pattern|value|param}} {{pattern|value|param|scope}} {{pattern}} {{nopattern}}	A reference to the specified pattern value. The OPTIONAL value argument specifies one of the pattern values for the referenced parameter. If present, it MUST be a valid pattern value for that parameter. The OPTIONAL param and scope arguments identify the referenced parameter. Scope defaults to normal. If present, param SHOULD adhere to the rules of A.2.3.4. If omitted, the current parameter is assumed. If the arguments are omitted, this is a hint to processing tools to replace the template reference with a list of the parameter's patterns, possibly preceded by text such as "Possible patterns:". This overrides processing tools' expected default behavior (unless suppressed by {{nopattern}}) of listing the parameter's patterns after the rest of the description. Otherwise, typically processing tools will (a) validate that the pattern value is valid, and (b) replace the template reference with the value and/or param arguments, appropriately formatted and with the value possibly rendered in a distinctive font. Processing tools can use the scope to convert a relative path into an absolute path in order, for example, to generate a hyperlink. Markup examples: {{pattern|None}} {{pattern|None|OtherParam}}

Name	Markup Definition	Description
Hidden value	{{hidden}} {{hidden|value}} {{nohidden}}	Text explaining that the value of the current parameter is hidden and cannot be read. This template SHOULD only be used within the description of a hidden parameter (A.2.7.1). This is a hint to processing tools to replace the template reference with text explaining that the value of the current parameter is hidden and cannot be read. This overrides processing tools' expected default behavior (unless suppressed by {{nohidden}}) of placing this text after the rest of the description. The OPTIONAL argument indicates the value that is returned when the current parameter is read. If omitted this defaults to the expansion of the {{null}} template. Typically, processing tools will generate text of the form "When read, this parameter returns <arg>, regardless of the actual value.".
Command parameter	{{command}} {{nocommand}}	Text explaining that the current parameter is a command parameter that triggers a CPE action. This template SHOULD only be used within the description of such a command parameter (A.2.7.1). This is a hint to processing tools to replace the template reference with text explaining that the current parameter is a command parameter that always reads back as {{null}}. This overrides processing tools' expected default behavior (unless suppressed by {{nocommand}}) of placing this text after the rest of the description. Typically, processing tools will generate text of the form "The value is not part of the device configuration and is always {{null}} when read.".
Factory default value	{{factory}} {{nofactory}}	Text listing the factory default for the current parameter. This template SHOULD only be used within the description of a parameter that has a factory default value. This is a hint to processing tools to replace the template reference with text listing the factory default value. This overrides processing tools' expected default behavior (unless suppressed by {{nofactory}}) of placing this text after the rest of the description. Typically, processing tools will generate text of the form "The factory default value MUST be <value>.".
Unique keys description	{{keys}} {{nokeys}}	A description of the current object's unique keys. This template SHOULD only be used within the description of a Multi-Instance Object (table) that defines one or more unique keys (A.2.8.1). This is a hint to processing tools to replace the template reference with a description of the object's unique keys. This overrides processing tools' expected default behavior (unless suppressed by {{nokeys}}) of describing the unique keys after the description.
Units reference	{{units}}	The parameter's units string. Typically, processing tools will (a) check that the parameter has a units string, and (b) substitute the value of its units string.
Boolean values	{{false}} {{true}}	Boolean values. Typically, processing tools will substitute the value False or True, possibly rendered in a distinctive font.
Miscellaneous	{{empty}}	Represents an empty string. Typically, processing tools will render such values in a distinctive font, possibly using standard wording, such as <Empty> or "an empty string".
	{{null}}	Expands to the appropriate null value for the current parameter's data type (A.2.3.5), e.g. {{empty}}, {{false}} or 0.

A.2.2.5 HTML Example

This includes examples of most of the markup and templates.

```
<model name="Goo:1.1" base="Goo:1.0">
  <object name="GooTop." access="readOnly" minEntries="1" maxEntries="1">
    <parameter name="ExampleParam" access="readOnly">
    <description>
      {{section|Introduction}}This is an ''example'' parameter that illustrates many of the
                  '''formatting''' templates.  For '''''example''''', this references {{bibref|TR-
                  106a1|section 3.2}}.
      {{section|Usage}}This parameter is called {{object}}{{param}}.  One can also reference other
                  parameters in the same object, such as {{param|OtherParameter}}, and indicate
                  that the parameter value is measured in {{units}}.
      One can also include bulleted lists:
      * level one
      ** level two
      * level one again
      ** level two again
      *** level three
      *: level one continued
      and numbered lists:
      # level one
      ## level two
      # level one again
      ## level two again
      ### level three
      #: level one continued
      and indented lists
      : level one
      :: level two
      : level one again
      :: level two again
      ::: level three
      and hyperlinks such as http://www.google.com
      and code examples:
        if (something) {
          /* do something */
        } else {
          /* do other */
        }
      If the parameter was Boolean, one could refer to its values {{false}} and {{true}}.
      One can refer to its enumerations individually, e.g. {{enum|Disabled}}, or to other parameters'
                  enumerations, such as {{enum|Value|OtherParam}}, or can list them all.  {{enum}}
      Finally, if there were any patterns they could be listed too.  {{pattern}}
    </description>
    <syntax>
      <string>
        <enumeration value="A"/>
        <enumeration value="B"/>
        <units value="packets"/>
      </string>
    </syntax>
  </parameter>
```

The resulting HTML would look something like this:

This is an *example* parameter that illustrates many of the **formatting** templates. For ***example***, this references [TR-106a1] section 3.2.

This parameter is called *ParentObject.ExampleParam*. One can also reference other parameters in the same object, such as *OtherParameter*, and indicate that the parameter value is measured in packets.

One can also include bulleted lists:

- level one
  - level two
- level one again
  - level two again
    - level three

  level one continued

and numbered lists:

1. level one
   1. level two
2. level one again
   1. level two again
      1. level three

   level one continued

and indented lists

level one
level two
level one again
level two again
level three

and hyperlinks such as http://www.google.com

and code examples:

```
if (something) {
  /* do something */
} else {
  /* do other */
}
```

If the parameter was Boolean, one could refer to its values *false* and *true*.

One can refer to its enumerations individually, e.g. *A*, or to other parameters' enumerations, such as *Value*, or can list them all. Possible values:

- *Disabled*
- *Enabled*
- *Error* (OPTIONAL)

Finally, if there were any patterns they could be listed too.

A.2.3 Data Types

TR-069 [2] data models support only the primitive data types listed in the last row of Table 4 “on the wire”. However, the DM Schema allows data types to be derived from the primitive types or from other named data types. Such derived types can be named or anonymous.

A.2.3.1 Named Data Types

Named data types are defined using the top-level dataType element. A DM Instance can contain zero or more top-level dataType elements.

When defining a new named data type, the following attributes and elements are relevant (normative requirements are specified in the schema).

Table 4 – XML Named Data Types

Name	Description
name	The data type name.
base	The base type name, i.e. name of the data type from which this data type is derived. This is used only where the base type is itself a named data type, not a primitive type.
status	The data type's {current, deprecated, obsoleted, deleted} status. This defaults to current, and so is not likely to be specified for a new data type.
description	The data type's description (A.2.2).
size pathRef instanceRef range enumeration enumerationRef pattern units	Data type facets (A.2.3.3). These are permitted only when the base type is a named data type, i.e. when the base attribute is specified.
base64 boolean dateTime hexBinary int long string unsignedInt unsignedLong	Primitive data type definition. These are permitted only when the base type is primitive. There is an element for each primitive data type, and each element supports only the facets (A.2.3.3) that are appropriate to that data type.

For example:

```
<dataType name="String255">
  <string>
    <size maxLength="255"/>
  </string>
</dataType>

<dataType name="String127" base="String255">
  <size maxLength="127"/>
</dataType>
```

A.2.3.2 Anonymous Data Types

Anonymous data types are defined within parameter syntax elements (A.2.7.1), and can apply only to the parameters within which they are defined. For example:

```
<parameter name="Example1" access="readOnly">
  <syntax>
    <string>
      <size maxLength="127"/>
    </string>
  </syntax>
</parameter>

<parameter name="Example2" access="readOnly">
  <syntax>
    <dataType base="String255">
      <size maxLength="127"/>
    </dataType>
  </syntax>
</parameter>
```

If an anonymous data type is modified in a later version of a data model, the modified anonymous data type is

regarded as being derived from the original anonymous data type. Therefore the base type restriction rules of A.2.3.8 MUST be obeyed.

A.2.3.3 Data Type Facets

A facet specifies some aspect of a data type, e.g. its size, range or units.

Note that XML Schema [11] also associates facets with data types. The XML Schema and DM Schema concepts are the same, but the set of facets is not identical.

The DM Schema defines the following facets (normative requirements are specified in the schema):

Table 5 – XML Data Type Facets

Name	Description
size	Size ranges for the data type (applies to string, base64, hexBinary and their derived types). Note that the size facet always refers to the actual value, not to the base64- or hexBinary-encoded value. Prior to the definition of the DM Schema, the maximum sizes of base64 parameters referred to the base64-encoded values. Processing tools that generate reports from DM Instances SHOULD include explicit clarification of whether the size ranges refer to the actual or encoded values. Note that the size facet is also used to specify the size range for list-valued parameters, which are always strings (A.2.7.1).
pathRef	Details of how to reference parameters and objects via their Path Names (applies to string and its derived types: A.2.3.7).
instanceRef	Details of how to reference object instances (table rows) via their Instance Numbers (applies to int, unsignedInt and their derived types; A.2.3.7).
range	Value ranges for the data type (applies to numeric data types and their derived types).
enumeration	Enumerations for the data type (applies to string and its derived types).
enumerationRef	Enumerations for the data type, obtained at run-time from the value of a specified parameter (applies to string and its derived types; A.2.3.7).
pattern	Patterns for the data type (applies to string and its derived types). Pattern value syntax is the same as for XML Schema regular expressions. See [14] Section F.
units	Units for the data type (applies to numeric data types and their derived types).

It is important to note that the enumeration facet does not necessarily define all the valid values for a data type. This is for the following reasons:

- As specified in Section 3.3, vendors are allowed to add additional enumeration values.
- A future version of a data model may need to add additional enumerations values.

A.2.3.4 Reference Path Names

Some description templates (A.2.2.4), and all reference facets (A.2.3.7), need to specify parameter or object names. It is always possible to specify a full Path Name, but it is frequently necessary or convenient to specify a relative Path Name. For example, it might be necessary to reference another parameter in the current object. Any Instance Numbers in the parameter's full Path Name cannot be known at data model definition time, so this can only be done using a relative Path Name.

The following rules apply to all Path Names that are used in data model definitions for referencing parameters or objects:

- Path Names MAY contain "{i}" placeholders, which MUST be interpreted as wild cards matching all Instance Numbers, e.g. "InternetGatewayDevice.WANDevice.{i}." references all WANDevice instances.
- Path Names MUST NOT contain Instance Numbers.

A Path Name is always associated with a path name scope, which defines the point in the naming hierarchy relative to which the Path Name applies.

Table 6 – Path Name Scope Definition

Name	Description
normal	This is a hybrid scope which usually gives the desired behavior: • If the path begins with a "Device" or "InternetGatewayDevice" component, it is relative to the top of the naming hierarchy. • If the path begins with a dot, it is relative to the Root or Service Object (c.f. scope=model). • Otherwise, the path is relative to the current object (c.f. scope=object).
model	The path is relative to the Root or Service Object.
object	The path is relative to the current object.

Formally, if the path name scope is normal:

- If the path is empty, it MUST be regarded as referring to the top of the naming hierarchy.
- Otherwise, if the path begins with a "Device" or "InternetGatewayDevice" component, it MUST be regarded as a full Path Name (these are the only two possible Root Device names).
- Otherwise, if the path begins with a dot ("."), it MUST be regarded as a path relative to the Root or Service Object. For example, in the Device Root Object ".DeviceInfo." means "Device.DeviceInfo.", and in the Device.Services.ABCService.1 Service Object it means "Device.Services.ABCService.1.DeviceInfo.".
- Otherwise, it MUST be regarded as a path relative to the current object. Any leading hash characters ("#") cause it to be relative to the parent of the current object (or the parent's parent, and so on) as described below. For example, if the current object is "Device.LAN.", "IPAddress" means "Device.LAN.IPAddress", "Stats." means "Device.LAN.Stats." and "#.DeviceInfo." means "Device.DeviceInfo" (see below for more "#" examples).

If the path name scope is model:

- If the path is empty, it MUST be regarded as referring to the Root or Service Object.
- Otherwise, it MUST be regarded as a path relative to the Root or Service Object. Any leading dot MUST be ignored. Leading hash characters are not permitted.

If the path name scope is object:

- If the path is empty, it MUST be regarded as referring to the current object.
- Otherwise, it MUST be regarded as a path relative to the current object. Any leading dot MUST be ignored. Leading hash characters are not permitted.

As mentioned above, if the path name scope is normal, a leading hash character causes the path to be relative to the parent of the current object. Additional hash characters reference the parent's parent, and so on, but they MUST NOT be used to reference beyond the Root or Service Object. Also, for object instances, "#." always means the Multi-Instance Object's (table's) parent rather than the Multi-Instance Object (table).

In addition, within a component definition, items that are defined outside the component MUST NOT be referenced via relative paths. This is because components can be included anywhere within the data model tree.

For example, if the current object is "Device.LAN.DHCPOption.{i}.":

- "#." means "Device.LAN." (the table's parent, not the table).
- "#.DHCPOption." means "Device.LAN.DHCPOption." (the table).
- "#.Stats." means "Device.LAN.Stats.".
- "#.Stats.TotalBytesSent" means "Device.LAN.Stats.TotalBytesSent".

The following examples would be invalid if LAN was defined within a component:

- "##." means "Device.".
- "##.DeviceInfo." means "Device.DeviceInfo.".
- "##.DeviceInfo.Manufacturer" means "Device.DeviceInfo.Manufacturer".

The final example can never be valid:

- "###." is not permitted (references beyond the Root Object).

Note that the term "Root or Service Object", which is used several times above, means "if within a Service Object instance, the Service Object instance; otherwise, the Root Object".

For example, the pathRef and instanceRef facets (A.2.3.7) have a targetParent attribute which specifies the possible parent(s) of the referenced parameter or object, and a targetParentScope attribute (defaulted to normal) which specifies targetParent's scope. If the current object is within a Service Object instance, setting targetParentScope to model forces the referenced parameter or object to be in the same Service Object instance. Similarly, setting targetParentScope to object forces the referenced parameter or object to be in the same object or in a sub-object.

A.2.3.5 Null Values and References

Each primitive data type has an associated null value that is used, for example, as the expansion of the {{null}} template (A.2.2.4). These null values are defined as follows:

- **base64, hexBinary, string:** an empty string
- **unsignedInt, unsignedLong:** 0
- **int, long:** -1
- **boolean:** false
- **dateTime:** 0001-01-01T00:00:00Z (the Unknown Time; see Section 3.2.2)

A null reference indicates that a reference parameter is not currently referencing anything. The value that indicates a null reference is the null value for the reference parameter's base data type, i.e.:

- **string**: an empty string
- **unsignedInt**: 0
- **int**: -1

A.2.3.6 Reference Types

A reference to another parameter or object can be weak or strong:

- **weak**: it does not necessarily reference an existing parameter or object. For example, if the referenced parameter or object is deleted, the value of the reference parameter might not get updated. All weak reference parameters MUST be declared as writable.
- **strong**: it always either references a valid parameter or object, or else is a null reference (A.2.3.5). If the referenced parameter or object is deleted, the value of the reference parameter is always set to a null reference.

See Section 3.2.4 for normative requirements relating to reference types and the associated CPE behavior.

A.2.3.7 Reference Facets

A reference facet specifies how a parameter can reference another parameter or object. There are three sorts of reference:

- **Path reference**: references another parameter or object via its Path Name. Details are specified via the pathRef facet, which applies to string and its derived types.
- **Instance reference**: references an object instance (table row) via its Instance Number. Details are specified via the instanceRef facet, which applies to int, unsignedInt and their derived types.
- **Enumeration reference**: references a list-valued parameter via its Path Name. The current value of the referenced parameter indicates the valid enumerations for this parameter. Details are specified via the enumerationRef facet, which applies to string and its derived types.

When defining a path reference, the following attributes and elements are relevant (normative requirements are specified in the schema).

Table 7 – PathRef Facet Definition

Name	Description
targetParent	An XML list of Path Names that can restrict the set of parameters and objects that can be referenced. If the list is empty (the default), then anything can be referenced. Otherwise, only the immediate children of one of the specified objects can be referenced, A "{i}" placeholder in a Path Name acts as a wild card, e.g. "InternetGatewayDevice.WANDevice.{i}.WANConnectionDevice.{i}.WANPPPConnection.". Path Names cannot contain explicit Instance Identifiers.
targetParentScope	Specifies the point in the naming hierarchy relative to which targetParent applies (A.2.3.4): normal (default), model or object.
targetType	Specifies what types of item can be referenced: • **any**: any parameter or object can be referenced (default) • **parameter**: any parameter can be referenced • **object**: any object can be referenced • **single**: any single-instance object can be referenced • **table**: any Multi-Instance Object (table) can be referenced • **row**: any Multi-Instance Object (table) instance (row) can be referenced

Name	Description

targetDataType	Specifies the valid data types for the referenced parameter. Is relevant only when targetType is any or parameter.
	Possible values are as follows:
	• **any**: a parameter of any data type can be referenced (default)
	• **base64**: only a base64 parameter can be referenced
	• **boolean**: only a boolean parameter can be referenced
	• **dateTime**: only a dateTime parameter can be referenced
	• **hexBinary**: only a hexBinary parameter can be referenced
	• **integer**: only an integer (int, long, unsignedInt or unsignedLong) parameter can be referenced
	• **int**: only an int parameter can be referenced
	• **long**: only a long (or int) parameter can be referenced
	• **string**: only a string parameter can be referenced
	• **unsignedInt**: only an unsignedInt parameter can be referenced
	• **unsignedLong**: only an unsignedLong (or unsignedInt) parameter can be referenced
	• **<named data type>**: only a parameter of the named data type can be referenced
	In addition, a parameter whose data type is derived from the specified data type can be referenced. The built-in type hierarchy (a simplified version of the XML Schema type hierarchy) is as follows:
	`any`
	`  base64`
	`  boolean`
	`  dateTime`
	`  hexBinary`
	`  integer`
	`  long`
	`  int`
	`  unsignedLong`
	`  unsignedInt`
	`  string`
	Note that *any* and *integer* are not valid parameter data types. They are included in order to support "can reference any data type" and "can reference any numeric data type".

refType	Specifies the reference type (A.2.3.6): weak or strong.

When defining an instance reference, the following attributes and elements are relevant (normative requirements are specified in the schema).

Table 8 – InstanceRef Facet Definition

Name	Description
targetParent	Specifies the Path Name of the Multi-Instance Object (table) of which an instance (row) is being referenced. "{i}" placeholders and explicit Instance Identifiers are not permitted in the Path Name. targetParentScope can be used to specify Path Names relative to the Root or Service Object or the current object.
targetParentScope	Specifies the point in the naming hierarchy relative to which targetParent applies (A.2.3.4): normal (default), model or object.
refType	Specifies the reference type (A.2.3.6): weak or strong.

When defining an enumeration reference, the following attributes and elements are relevant (normative requirements are specified in the schema).

Table 9 – EnumerationRef Facet Definition

Name	Description
targetParam	Specifies the Path Name of the list-valued parameter whose current value indicates the valid enumerations for this parameter.
targetParamScope	Specifies the point in the naming hierarchy relative to which targetParam applies (A.2.3.4): normal (default), model or object.

Name	Description
nullValue	Specifies the parameter value that indicates that none of the values of the referenced parameter currently apply (if not specified, no such value is designated). Note that if this parameter is list-valued then nullValue is not relevant, because this condition will be indicated by an empty list.

The following examples illustrate the various possible types of reference.

```
    <object name="PeriodicStatistics.SampleSet.{i}.Parameter.{i}." …>
      …
      <parameter name="Reference" access="readWrite">
        <description>Reference to the parameter that is associated with this object instance.
This MUST be the parameter's full path name.</description>
        <syntax>
          <string>
            <size maxLength="256"/>
            <pathRef targetType="parameter" refType="weak"/>
          </string>
          <default type="object" value=""/>
        </syntax>
      </parameter>
```

```
    <object name="StorageService.{i}.StorageArray.{i}." …>
      …
      <parameter name="PhysicalMediumReference" access="readWrite">
        <description>A comma-separated list of Physical Medium references.  Each Physical Medium
                     referenced by this parameter MUST exist within the same StorageService instance.
                     A Physical Medium MUST only be referenced by one Storage Array instance. Each
                     reference can be either in the form of ".PhysicalMedium.{i}" or a fully
                     qualified object name…</description>
        <syntax>
          <list>
            <size maxLength="1024"/>
          </list>
          <string>
            <pathRef targetParent=".PhysicalMedium." targetParentScope="model" targetType="row"
                     refType="strong"/>
          </string>
        </syntax>
      </parameter>
```

```
    <object name="InternetGatewayDevice.QueueManagement.Classification.{i}." access="readWrite"
                     minEntries="0" maxEntries="unbounded"
                     entriesParameter="ClassificationNumberOfEntries">
      <description>Classification table.</description>
      <parameter name="ClassQueue" access="readWrite">
        <description>Classification result.  Instance Number…</description>
        <syntax>
          <int>
            <instanceRef targetParent=".QueueManagement.Queue." refType="strong"/>
          </int>
        </syntax>
      </parameter>
```



```
    <object name="STBService.{i}.Components.FrontEnd.{i}.IP.Inbound.{i}." …>
      …
      <parameter name="StreamingControlProtocol" access="readOnly">
        <description>Network protocol currently used for controlling streaming of the source content, or
                     an empty string if the content is not being streamed or is being streamed but is
                     not being controlled.
If non-empty, the string MUST be one of the .Capabilities.FrontEnd.IP.StreamingControlProtocols
                     values.</description>
        <syntax>
          <string>
            <enumerationRef targetParam=".Capabilities.FrontEnd.IP.StreamingControlProtocols"
                     nullValue=""/>
          </string>
        </syntax>
      </parameter>

      <parameter name="StreamingTransportProtocol" access="readOnly">
        <description>Network protocol currently used for streaming the source content, or an empty
                     string if the content is not being streamed.
If non-empty, the string MUST be one of the .Capabilities.FrontEnd.IP.StreamingTransportProtocols
                     values.</description>
        <syntax>
          <string>
            <enumerationRef targetParam=".Capabilities.FrontEnd.IP.StreamingTransportProtocols"
                     nullValue=""/>
          <string/>
        </syntax>
      </parameter>
```

```
    <object name="InternetGatewayDevice.LANDevice.{i}.WLANConfiguration.{i}.WPS." …>
      …
      <parameter name="ConfigMethodsEnabled" access="readWrite">
        <description>Comma-separated list of the WPS configuration methods enabled on the device.  Each
                     entry in the list MUST be a member of the list reported by the
                     ConfigMethodsSupported parameter…</description>
        <syntax>
          <list/>
          <string>
            <enumerationRef targetParam="ConfigMethodsSupported"/>
          </string>
        </syntax>
      </parameter>
```

A.2.3.8 Base Type Restriction

A new data type MUST always be a restriction of its base type, meaning that a valid value of the new data type will always be a valid value for its base type. This is the case for the examples of A.2.3.1, which involve three different data types:

- string of unlimited length
- string of maximum length 255
- string of maximum length 127

Clearly a string of length 100 is valid for all three data types, but a string of length 200 is only valid for the first two data types.

The examples of A.2.3.1 considered only the size facet, but in general all facets that are applicable to the data type have to be considered. The base type restriction requirements for each facet are as follows:

Table 10 – XML Facet Inheritance Rules

Facet	Requirements
size	The derived data type can define sizes in any way, provided that the new sizes do not permit any values that are not valid for the base type.

Facet	Requirements
pathRef	The derived data type can modify the data type in the following ways: • By "promoting" status to a "higher" value, where the lowest to highest ordering is: current, deprecated, obsoleted, deleted. For example, current can be changed to deprecated, and obsoleted can be changed to deleted, but deleted cannot be changed back to obsoleted. When promoting status, the deprecation, obsoletion and deletion rules of Section 2.4 MUST be obeyed. • By changing targetParent to narrow the set of possible parent objects. • By changing targetType to narrow the set of possible target types. • By changing targetDataType to narrow the set of possible target data types.
instanceRef	The derived data type can modify the data type in the following ways: • By "promoting" status to a "higher" value, as described for pathRef. • By changing targetParent to narrow the set of possible parent objects.
range	The derived data type can define ranges in any way, provided that the new ranges do not permit any values that are not valid for the base type.
enumeration	The derived data type can modify existing enumeration values in the following ways: • By "promoting" access from readOnly to readWrite. • By "promoting" status to a "higher" value, as described for pathRef. • By "promoting" optional from False to True. • By adding a code, if none was previously specified. • By using the action attribute to extend or replace the description (see below). The derived data type can add new enumeration values.
enumerationRef	The derived data type can modify the data type in the following ways: • By "promoting" status to a "higher" value, as described for pathRef.
pattern	The derived data type can modify existing pattern values by changing access, status, optional and description exactly as for enumerations. The derived data type can add new patterns and/or replace existing patterns with new patterns, provided that the new patterns do not permit any values that are not valid for the base type. For example a single pattern "[AB]" could be replaced with "A" and "B", but "C" could not be added.
units	The derived data type can add units if the base type did not specify any.

Most of the above requirements are non-normative, because it has to be possible to correct errors. For example, if the base type supports a range of [-1:4095] but the values 0 and 4095 were included in error, it would be permissible for a derived type to support ranges of [-1:-1] and [1:4094]. Processing tools SHOULD be able to detect and warn about such cases.

When defining a new data type, if a facet is omitted, the new data type will inherit that facet from its base type. If a facet is present, it MUST be fully specified (except that special rules apply to descriptions; see below). For example, this means that a derived type that adds additional enumeration values has also to re-declare the enumeration values of the base type.

For example, in the following, the derived type inherits the units facet from its parent but it does not inherit the range facet, so the PacketCounter range is [10:] and the PacketCounter2 range is [15:20].

```
<dataType name="PacketCounter">
  <unsignedLong>
    <range minInclusive="10"/>
    <units value="packets"/>
  </unsignedLong>
</dataType>

<dataType name="PacketCounter2" base="PacketCounter">
  <range minInclusive="15" maxInclusive="20"/>
</dataType>
```

Similarly, in the following, the enumeration values for ABCD are not A, B, C and D, but are just C and D. This is an error (because the derived type cannot remove enumeration values), and processing tools SHOULD detect and warn about such cases.


```
<dataType name="AB">
  <string>
    <enumeration value="A"/>
    <enumeration value="B"/>
  </string>
</dataType>

<dataType name="ABCD" base="AB">
  <string>
    <enumeration value="C"/>
    <enumeration value="D"/>
  </string>
</dataType>
```

A derived data type and any of its facets that support descriptions will inherit those descriptions from the base type. Facet descriptions are inherited regardless of whether the facet is present in the derived type. For any descriptions that are explicitly specified in the derived type, the action attribute controls whether they will be extended or replaced.

For example, in the following, the description of Z (which is not changed) does not have to be repeated.

```
<dataType name="XY">
  <description>This is XY.</description>
  <string>
    <enumeration value="X">
      <description>This is X.</description>
    </enumeration>
    <enumeration value="Y">
      <description>This is Y.</description>
    </enumeration>
    <enumeration value="Z">
      <description>This is Z.</description>
    </enumeration>
  </string>
</dataType>

<dataType name="XY2" base="XY">
  <description action="replace">This is all about XY.</description>
  <enumeration value="X">
    <description action="append">This is more about X, added at the end.</description>
  </enumeration>
  <enumeration value="Y">
    <description action="prefix">This is more about Y, inserted at the beginning.</description>
  </enumeration>
  <enumeration value="Z"/>
</dataType>
```

A.2.4 Bibliography

The bibliography is defined using the top-level bibliography element, which can contain zero or more (bibliographic) reference elements.

When defining a new bibliographic reference, the following attributes and elements are relevant (normative requirements are specified in the schema).

Table 11 – XML Bibliographic References

Name	Description
id	The bibliographic reference ID.
name	The name by which the referenced document is usually known.
title	The document title.
organization	The organization that published the referenced document, e.g. BBF, IEEE, IETF.
category	The document category, e.g. TR (BBF), RFC (IETF).
date	The publication date.
hyperlink	Hyperlink(s) to the document.

The bibliographic reference ID is intended to uniquely identify this reference across all instance documents. Therefore, for instance documents that are published by the BBF, IDs MUST obey the following rules:

- For a BBF Technical Report, it MUST be of the form "TR-nnnixaycz", where TR is the literal "TR", nnn is the Technical Report number (including leading zeros), i, a and c are literal letters, and x, y, and z are the issue, amendment and corrigendum numbers respectively. The issue number (ix) is omitted if it is issue 1 and the amendment number (ay) is omitted if it is amendment 0. For example, "TR-106a2" refers to TR-106 (Issue 1) Amendment 2. If the corrigendum number (cz) is omitted, the most recent corrigendum is assumed.
- For an IETF RFC, it MUST be of the form "RFCnnn", where RFC is the literal "RFC" and nnn is the RFC number (no leading zeros).
- For an IEEE specification, it SHOULD be of the form "nnn.ml-dddd", where nnn.m is the IEEE group, l is the spec letter(s), and dddd is the publication year. For example, "802.1D-2004".
- For an ETSI specification (which includes DVB specifications), it SHOULD be of the form "TTnnnnnnva.b.c" where TT is the specification type, usually "TS" (Technical Specification), nnnnnn is the specification number, and a.b.c is the version number.
- For specifications issued by other standards organizations, or by vendors, it SHOULD be of a standard form if one is defined.

Processing tools SHOULD be lenient when comparing bibliographic reference IDs. Specifically, they SHOULD ignore all whitespace, punctuation, leading zeros in numbers, and upper / lower case. So, for example, "rfc 1234" and "RFC1234" would be regarded as the same ID, as would "TR-069" and "TR69".

Processing tools SHOULD detect and report inconsistent bibliographic references, e.g. a reference with the same ID (i.e. an ID that compares as equal) as one that was encountered in a different file, but with a different name or hyperlink.

Formally, bibliographic reference IDs in instance documents that are published by the BBF and the other organizations mentioned above are defined as follows:


```
ReferenceID = BBFID
            | RFCID
            | IEEEID
            | ETSIID
            | OtherID

BBFID = "TR-" BBFNumber BBFIssue BBFAmendment BBFCorrigendum

BBFNumber = [DIGIT]{3,}     // including leading zeros, e.g. 069

BBFIssue = "i" <number greater than one>
         | ""               // empty means Issue 1

BBFAmendment = "a" <number greater than zero>
             | ""           // empty means Amendment 0

BBFCorrigendum = "c" <number greater than zero>
               | ""         // empty means the most recent Corrigendum

RFCID = "RFC" RFCNumber

RFCNumber = NONZERODIGIT [DIGIT]*
                            // no leading zeros, e.g. 123

IEEEID = IEEEGroup IEEESpec IEEEDate
       | <for other IEEE specifications, of a standard form if one is defined>

IEEEGroup = <group number> "." <group sub-number>
                            // e.g. 802.1

IEEESpec = <spec letter(s)>     // e.g. D

IEEEDate = "-" <publication year>
                            // e.g. -2004
         | ""               // can be empty

ETSIID = ETSISpecType ETSINumber ETSIVersion
       | <for other ETSI specifications, of a standard form if one is defined>

ETSISpecType = "TR"         // Technical Report
             | "TS"         // Technical Specification
             | "ES"         // ETSI Specification
             | "EN"         // European Standard

ETSINumber = [DIGIT]{6}     // e.g. 102034

ETSIVersion = "v" <version number as specified by ETSI>
            | ""            // can be empty

OtherURI = <of a standard form if one is defined>
```

A.2.5 Components

A component is a way of defining a named set of parameters, objects and/or profiles to be used wherever such a group is needed in more than one place (or just to structure the definitions). A DM Instance can contain zero or more top-level component elements.

When defining a new component, the following attributes and elements are relevant (normative requirements are specified in the schema).

Table 12 – XML Component Definition

Name	Description
name	The component name.
description	The component's description (A.2.2).
component	The other components that are referenced (included) by this component.
parameter	The component's top-level parameter definitions (A.2.7).

Name	Description
object	The component's object definitions (A.2.8).
profile	The component's profile definitions (A.2.9).

Referencing (including) a component can be thought of as textual substitution. A component has no version number and is not tied to a particular Root or Service Object.

The following is a simple example of component definition and reference.

```
<component name="ByteStats">
  <parameter name="BytesSent" access="readOnly">
    <description>Number of bytes sent.</description>
    <syntax><unsignedInt/></syntax>
  </parameter>
  <parameter name="BytesReceived" access="readOnly">
    <description>Number of bytes received.</description>
    <syntax><unsignedInt/></syntax>
  </parameter>
</component>

<model name="InternetGatewayDevice:1.4">
  <object name="InternetGatewayDevice." access="readOnly" minEntries="1" maxEntries="1">
    …
    <component ref="ByteStats"/>
    …
  </object>
  …
</model>
```

Here the component is referenced from within an object definition. Components can be referenced from within component, model and object definitions. Parameter, object and profile definitions within components are relative to the point of inclusion unless overridden using the path attribute.

A.2.6 Root and Service Objects

Root and Service Objects are defined using the model element and an associated top-level object element. A DM Instance can contain zero or more top-level model elements.

When defining a new model, the following attributes and elements are relevant (normative requirements are specified in the schema).

Table 13 – XML Root and Service Objects

Name	Description
name	The model name, including its major and minor version numbers (3.7).
base	The name of the previous version of the model (for use when the model version is greater than 1.0).
isService	Whether it is a Service Object. This defaults to False and so can be omitted for Root Objects.
description	The model's description (A.2.2).
component	The components that are referenced (included) by the model.
parameter	The model's top-level parameter definitions (A.2.7).
object	The model's top-level and other object definitions (A.2.8).
profile	The model's profile definitions (A.2.9).

Once a given version has been defined, it cannot be modified; instead, a new version of the object has to be defined. For example, the following example defines v1.0 and v1.1 of a notional Service Object.


```
<model name="DemoService:1.0" isService="true">
  <parameter name="DemoServiceNumberOfEntries" access="readOnly"/>
  <object name="DemoService.{i}." access="readOnly" minEntries="0" maxEntries="unbounded"
                                          entriesParameter="DemoServiceNumberOfEntries"/>
</model>

<model name="DemoService:1.1" base="DemoService:1.0" isService="true">
  <object base="DemoService.{i}." access="readOnly" minEntries="0" maxEntries="unbounded"/>
</model>
```

A.2.7 Parameters

Parameters are defined using the parameter element, which can occur within component, model and object elements. When defining a new parameter, the following attributes and elements are relevant (normative requirements are specified in the schema).

Table 14 – XML Parameter Definition

Name	Description
name	The parameter name (3.1).
access	Whether the parameter can be writable (readWrite) or not (readOnly).
status	The parameter's {current, deprecated, obsoleted, deleted} status. This defaults to current, and so is not likely to be specified for a new parameter.
activeNotify	The parameter's {normal, forceEnabled, ForceDefault, canDeny} Active Notification status. This defaults to normal, and so is not often specified for a new parameter.
forcedInform	The parameter's Forced Inform status. This defaults to False, and so is not often specified for a new parameter.
description	The parameter's description (A.2.2).
syntax	The parameter's syntax (A.2.7.1).

A.2.7.1 Parameter Syntax

Parameter syntax is defined using the syntax element, which can occur only within parameter elements. When defining a new parameter, the following attributes and elements are relevant (normative requirements are specified in the schema).

Table 15 – XML Parameter Syntax

Name	Description
hidden	Whether the value is hidden on readback. This defaults to False, and so is not often specified for a new parameter.
command	Whether setting the parameter triggers a CPE action. This defaults to False, and so is not often specified for a new parameter.
list	If the parameter is list-valued, details of the list value. This allows specification of the maximum and minimum number of items in the list, and also supports a size facet for the list (A.2.3.3). Note that a list-valued parameter is always a string as far as TR-069 [2] is concerned. For a list, the rest of the syntax specification refers to the individual list items, not to the parameter value.
base64 boolean dateTime hexBinary int long string unsignedInt unsignedLong	If the parameter is of a primitive data type, specifies a primitive data type reference, e.g. <int/>. If the parameter data type is derived from a primitive data type, specifies an anonymous primitive data type definition (A.2.3.2), e.g. <int><range maxInclusive="255"/></int>. Each primitive data type element supports only the facets (A.2.3.3) that are appropriate to that data type.
dataType	If the parameter is of a named data type, specifies a named data type (A.2.3.1) reference, e.g. <dataType ref="IPAddress"/>. If the parameter data type is derived from a named data type, specifies an anonymous named data type (A.2.3.2) definition, e.g. <dataType base="IPAddress"><size maxLength="15"/></dataType>.

A.2.8 Objects

Objects are defined using the object element, which can occur within component and model elements. When defining a new object, the following attributes and elements are relevant (normative requirements are specified in the schema).

Table 16 – XML Object Definition

Name	Description
name	The object name, specified as a partial path (3.1).
access	Whether object instances can be Added or Deleted (readWrite) or not (readOnly). Adding or deleting instances is meaningful only for a Multi-Instance Object (table).
minEntries	The minimum number of instances of this object (always less than or equal to maxEntries).
maxEntries	The maximum number of instances of this object (can be "unbounded"). minEntries and maxEntries allow the object to be placed into one of three categories: • **minEntries=0, maxEntries=1:** single-instance object which might not be allowed to exist, e.g. because only one of it and another object can exist at the same time. • **minEntries=1, maxEntries=1:** single-instance object that is always allowed to exist. • **All other cases:** Multi-Instance Object (table) (A.2.8.1).
status	The object's {current, deprecated, obsoleted, deleted} status. This defaults to current, and so is not likely to be specified for a new object.
description	The object's description (A.2.2).
component	The components that are referenced (included) by the object.
parameter	The object's parameter definitions (A.2.7).

A.2.8.1 Tables

If an object is a table, several other attributes and elements are relevant (normative requirements are specified in the schema).

Table 17 – XML Table Definition

Name	Description
name	For a table, the last part of the name has to be "{i}" (3.1).
entriesParameter	The name of the parameter (in the parent object) that contains the number of entries in the table. Such a parameter is needed whenever there is a variable number of entries, i.e. whenever maxEntries is unbounded or is greater than minEntries.
enableParameter	The name of the parameter (in each table entry) that enables and disables that table entry. Such a parameter is needed whenever access is readWrite (so the ACS might be able to create entries) and at least one uniqueKey element that defines a functional key is present.
uniqueKey	An element that specifies a unique key by referencing those parameters that constitute the unique key. For a non-functional key, or if the table has no enableParameter, the uniqueness requirement always applies. For a functional key, and if the table has an enableParameter, the uniqueness requirement applies only to enabled table entries.

Each unique key is either functional or non-functional:

- A functional key references at least one parameter that relates to the purpose (or function) of the table, e.g. a DHCP option tag in a DHCP option table, or an external port number in a port mapping table.
- A non-functional key references only parameters that do not relate to the purpose (or function) of the table, e.g. an Alias or Name parameter.

A unique key is assumed to be functional unless explicitly marked as non-functional by setting the unique key's functional attribute to false.

As can be seen from the description in Table 17, non-functional keys are always required to be unique, regardless of whether the table has an enableParameter, or is enabled or disabled. Therefore, at most one entry in a given parent object can exist with a given value for a non-functional unique key.

The uniqueness requirement means that the value of the unique key MUST be unique for all instances of a given parent object regardless of how instances got created.

A.2.9 Profiles

Profiles are defined using the profile element, which can occur within component and model elements. When defining a new profile, the following attributes and elements are always relevant (normative requirements are specified in the schema).

Table 18 – XML Profile Definition

Name	Description
name	The profile name, including its version number (2.3.3).
base	The name of the previous version of the profile (for use when the profile version is greater than 1).
extends	A list of the names of the profiles that this profile extends.
description	The profile's description (A.2.2).
parameter	The profile's parameter requirements, which can include descriptions, references to the parameters in question, and the parameter access requirement.
object	The profile's object requirements, which can include descriptions, references to the objects in question, the object access requirements, and requirements for the object's parameters.

A.2.10 Modifications

New data types, components, models and profiles can be created based on existing items. This does not modify the existing item.

Parameters, objects and profiles can be modified "in place", i.e. without creating a new item. This still uses the parameter, object and profile elements, and is indicated by using the base, rather than the name, attribute. The base attribute specifies the name of the existing item that is to be modified.

The syntax for modifying an item is the same as for creating an item, but there are rules. These rules are not specified in the DM Schema.

A.2.10.1 Parameter Modifications

The following rules govern parameter modifications.

Table 19 – XML Parameter Modification

Name	Description
access	Can be "promoted" from readOnly to readWrite.
status	Can be "promoted" to a "higher" value, where the lowest to highest ordering is: current, deprecated, obsoleted, deleted. For example, current can be changed to deprecated, and obsoleted can be changed to deleted, but deleted cannot be changed back to obsoleted. When promoting status, the deprecation, obsoletion and deletion rules of Section 2.4 MUST be obeyed.
activeNotify	Can be changed from forceEnabled to forceDefaultEnabled. No other changes are permitted.
forcedInform	Cannot be changed.
description	Can be extended or replaced via use of the action attribute. When changing the description, behavioral backwards compatibility MUST be preserved.
syntax/hidden	Cannot be changed.
syntax/list	Can add or modify the list element in the following ways: • Can convert a non-list string parameter to a list provided that an empty string was already a valid value with the appropriate meaning. • Can adjust limits on numbers of items, and on the list size, provided that the new rules do not permit any values that were not valid for the previous version of the parameter.
syntax/int etc syntax/dataType	Can make any change that follows the base type restriction rules of A.2.3.8, e.g. can add enumerations.
syntax/default	A default can be added if the parameter did not already have one.

Most of the above requirements are non-normative, because it has to be possible to correct errors in a previous version of a parameter. Processing tools SHOULD be able to detect and warn when a parameter is modified in a way that contravenes the above rules.

A.2.10.2 Object Modifications

The following rules govern object modifications.

Table 20 – XML Object Modification

Name	Description
access	Can be "promoted" from readOnly to readWrite.
minEntries	Cannot be changed.
maxEntries	Cannot be changed.
entriesParameter	Cannot be changed, unless was previously missing, in which case can be added.
enableParameter	Cannot be changed, unless was previously missing, in which case can be added.
status	Can be "promoted" to a "higher" value, where the lowest to highest ordering is: current, deprecated, obsoleted, deleted. For example, current can be changed to deprecated, and obsoleted can be changed to deleted, but deleted cannot be changed back to obsoleted. When promoting status, the deprecation, obsoletion and deletion rules of Section 2.4 MUST be obeyed.
description	Can be extended or replaced via use of the action attribute. When changing the description, behavioral backwards compatibility MUST be preserved.
uniqueKey	Cannot be changed.
component	Can reference (include) new components.
parameter	Can add new parameters.

Most of the above requirements are non-normative, because it has to be possible to correct errors in a previous version of an object. Processing tools SHOULD be able to detect and warn when an object is modified in a way that contravenes the above rules.

A.2.10.3 Profile Modifications

The following rules govern profile modifications. They apply to the profile element, and to its nested parameter and object elements.

Table 21 – XML Profile Modification

Name	Description
status	Can be "promoted" to a "higher" value, where the lowest to highest ordering is: current, deprecated, obsoleted, deleted. For example, current can be changed to deprecated, and obsoleted can be changed to deleted, but deleted cannot be changed back to obsoleted. When promoting status, the deprecation, obsoletion and deletion rules of Section 2.4 MUST be obeyed.
description	Can be extended or replaced via use of the action attribute. When changing the description, behavioral backwards compatibility MUST be preserved.

Most of the above requirements are non-normative, because it has to be possible to correct errors in a profile. Indeed, since profiles are immutable, the only valid reason for changing a profile is to correct errors. Processing tools SHOULD be able to detect and warn when a profile is modified in a way that contravenes the above rules.

A.3 DM Schema

The normative version of the DM Schema can be found at http://www.broadband-forum.org/cwmp/cwmp-datamodel-1-4.xsd. Please be aware that a new version of the DM Schema might be published at any time, in which case the version referenced in this document would become out of date. Any conflict MUST be resolved in favor of the normative version on the web site.

Annex B. CWMP Device Type XML Schema

B.1 Introduction

The CWMP Device Type XML Schema [11], or DT Schema, is used for describing a device's supported data model.

DT Schema instance documents can contain the following:

- Imports (from DM Schema instance documents) of Root or Service Object definitions
- Declarations of which features of imported Root or Service Objects are supported

DT Schema instance documents cannot contain definitions of Root or Service Objects. All such definitions have to reside in DM Schema instance documents.

B.2 Normative Information

It is possible to create instance documents that conform to the DT Schema but nevertheless are not valid device type specifications. This is because it is not possible to specify all the normative device type specification requirements using the XML Schema language. Therefore, the schema contains additional requirements written using the usual normative language. Instance documents that conform to the DT Schema and meet these additional requirements are referred to as DT Instances.

The question of the location of the definitive normative information therefore arises. The answer is as follows:

- All the normative information in the main part of the document remains normative.
- The DT Schema, and the additional requirements therein, are normative. Some of these additional requirements are duplicated (for emphasis) in this Annex.
- The DT Schema references additional material in this Annex. Such material is normative.
- If the DT Schema conflicts with a normative requirement in the main part of the document, this is an error in the DT Schema, and the requirement in the main part of the document takes precedence.

B.2.1 Importing DM Instances

DM Instances are imported using the top-level import element, which differs from the DM Schema import element in that only data types and models can be imported (components cannot be imported because they are not used in DT Instances).

> *Note – the rules for importing DM Instances into DT Instances are consistent with those given in A.2.1 for importing DM Instances into other DM Instances. The only difference is an additional rule governing the use, when available, of the DT Instance URL.*

The DT Schema specifies that the DM Instance is located via the file attribute.

The rules governing the file attribute's value and its use for locating the DM Instance are as follows:

- It MUST be a URL adhering to RFC 3986 [7].
- If the URL includes a scheme, it MUST be http, https or ftp.

- If the URL includes an authority, it MUST NOT include credentials.
- For standard BBF DM Instances, the rules that apply to the filename part (final path segment) of the A.2.1.1 BBFURL MUST be applied to the filename part of this URL. This means that the corrigendum number can be omitted in order to refer to the latest corrigendum.
- If the URL is a relative reference, processing tools MUST apply their own logic, e.g. apply a search path. If a DT Instance URL is available, the relative reference MUST be interpreted relative to the DT Instance URL.

B.2.2 Features

The feature element provides a simple way for a DT Instance to indicate whether a given feature is supported. The current set of standard features is as follows:

Feature	Description
DNSClient	Device contains a DNS client.
DNSServer	Device contains a DNS server.
Firewall	Device contains a firewall.
IPv6	Device supports IPv6.
NAT	Device supports NAT.
Router	Device is a router.

Vendor-specific features MAY be supported, and if so the feature name MUST begin with X_<VENDOR>_, where <VENDOR> MUST be as defined in Section 3.3.

This example feature declaration illustrates the use of annotation:

```
<feature name="DNSServer">
  <annotation>Supports a DNS Server and XYZ.</annotation>
</feature>
```

In order to make it easy to add new features, standard feature names are defined in a separate DT Features Schema that is imported by the DT Schema. The DT Features Schema is unversioned, so the DT Schema need not be changed when new standard feature names are added. In order to preserve backwards compatibility, standard feature names, once added, MUST NOT ever be deleted.

B.3 DT Features Schema

The normative version of the DT Features Schema can be found at http://www.broadband-forum.org/cwmp/cwmp-devicetype-features.xsd. Please be aware that a new version of the DT Features Schema might be published at any time, in which case the version referenced in this document would become out of date. Any conflict MUST be resolved in favor of the normative version on the web site.

B.4 DT Schema

The normative version of the DT Schema can be found at http://www.broadband-forum.org/cwmp/cwmp-devicetype-1-1.xsd. Please be aware that a new version of the DT Schema might be published at any time, in which case the version referenced in this document would become out of date. Any conflict MUST be resolved in favor of the normative version on the web site.

Appendix I. HTML Data Model Reports

I.1 Introduction

TR-106 Amendment 2 published the first version of the DM Schema. Since then the normative definitions of all TR-069 data models have been published as DM Instances (XML documents that conform to the DM Schema). Since these XML data models might not be easily read by a human, corresponding non-normative HTML data model reports have also been published (see http://www.broadband-forum.org/cwmp.php#DMD).

This appendix briefly discusses these HTML reports.

I.2 Report Types

There are two types of HTML reports published for a given version of a data model:

- a full report, covering the given version and all preceding versions of the data model
- a partial report, covering only the given version of the data model (i.e. excluding content specific to earlier versions of the data model); i.e. last only changes

For example, TR-181 Issue 1 defined the Device:1.5 data model revision, declared in tr-181-1-0-0.xml. The full report is in tr-181-1-0-0.html, and includes the aggregate data model definitions from Device:1.5 and earlier (back to Device:1.0 inclusive). The partial report is in tr-181-1-0-0-last.html, and only includes the data model definitions added or changed by TR-181 Issue 1.

I.3 Report Layout

Each HTML data model report contains the following sections:

Notice	The legal notice, lifted from the top of the associated XML data model file.
Summary	Describes the reason for this data model version. This is lifted from the associated XML data model file's document description.
Table of Contents	Hyperlinks to the various sections within the report, as well as links to each Object and Profile definition within the report.
Data Types	Named data type definitions (i.e. not built-in types) that are used to define Parameters within the report. Each data type definition consists of name, type, and description.
References	Hyperlinks to external bibliography references cited by Object, Parameter, and Profile descriptions within the report.
Data model definition	Object and Parameter definitions. Which Object and Parameter definitions are included depends on whether it is a full or partial (last only) report.
Inform and Notification Requirements	Lists those Parameters within the report that are: forced inform parameters, forced active notification parameters, and parameters for which active notification can be denied.
Profile Definitions	Profile definitions, showing Object and Parameter requirements. Which Profiles are included depends on whether it is a full or partial (last only) report.

I.4 Data Model Definition

Parameters make use of a limited subset of the default SOAP data types [5]. The notation used to represent these types within the report is listed in the following table.

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, a minimum and maximum allowed length can be indicated using the form string(Min:Max), where Min and Max are the minimum and maximum string length in characters. If either Min or Max are missing, this indicates no limit, and if Min is missing the colon can also be omitted, as in string(Max). Multiple comma-separated ranges can be specified, in which case the string length will be in one of the ranges.
int	Integer in the range –2147483648 to +2147483647, inclusive. For some int types, a value range is given using the form int[*Min:Max*] or int[*Min:Max* step *Step*] where the *Min* and *Max* values are inclusive. If either *Min* or *Max* are missing, this indicates no limit. If *Step* is missing, this indicates a step of 1. Multiple comma-separated ranges can be specified, in which case the value will be in one of the ranges.
long	Long integer in the range –9223372036854775808 to 9223372036854775807, inclusive. For some long types, a value range is given using the form long[*Min:Max*] or long[*Min:Max* step *Step*], where the *Min* and *Max* values are inclusive. If either *Min* or *Max* are missing, this indicates no limit. If *Step* is missing, this indicates a step of 1. Multiple comma-separated ranges can be specified, in which case the value will be in one of the ranges.
unsignedInt	Unsigned integer in the range 0 to 4294967295, inclusive. For some unsignedInt types, a value range is given using the form unsignedInt[*Min:Max*] or unsigned[*Min:Max* step *Step*], where the *Min* and *Max* values are inclusive. If either *Min* or *Max* are missing, this indicates no limit. If *Step* is missing, this indicates a step of 1.Multiple comma-separated ranges can be specified, in which case the value will be in one of the ranges.
unsignedLong	Unsigned long integer in the range 0 to 18446744073709551615, inclusive. For some unsignedLong types, a value range is given using the form unsignedLong[*Min:Max*] or unsignedLong[*Min:Max* step *Step*], where the *Min* and *Max* values are inclusive. If either *Min* or *Max* are missing, this indicates no limit. If *Step* is missing, this indicates a step of 1. Multiple comma-separated ranges can be specified, in which case the value will be in one of the ranges.
boolean	Boolean, where the allowed values are “0” or “1” (or equivalently, “true” or “false”).
dateTime	The subset of the ISO 8601 date-time format defined by the SOAP dateTime type.
base64	Base64 encoded binary (no line-length limitation). A minimum and maximum allowed length can be indicated using the form base64(Min:Max), where Min and Max are the minimum and maximum length in characters before Base64 encoding. If either Min or Max are missing, this indicates no limit, and if Min is missing the colon can also be omitted, as in base64(Max). Multiple comma-separated ranges can be specified, in which case the length MUST be in one of the ranges.
hexBinary	Hex encoded binary. A minimum and maximum allowed length can be indicated using the form hexBinary(Min:Max), where Min and Max are the minimum and maximum length in characters before Hex Binary encoding. If either Min or Max are missing, this indicates no limit, and if Min is missing the colon can also be omitted, as in hexBinary(Max). Multiple comma-separated ranges can be specified, in which case the length MUST be in one of the ranges.

Note: A Parameter that is defined to be one of the named data types, is reported as such at the beginning of the Parameter’s description via a reference back to the associated data type definition (e.g. [*MacAddress*]). However, such parameters still indicate their SOAP data type (as discussed in the table above).

End of Broadband Forum Technical Report TR-106

