TECHNICAL REPORT

TR-154

TR-069 Data Model XML User Guide

Issue: 1
Issue Date: March 2012

© The Broadband Forum. All rights reserved.

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
1	March 2012	Paul Sigurdson, Broadband Forum	Original

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

Editor	Paul Sigurdson	Broadband Forum
BroadbandHome™ Working Group Chairs	Greg Bathrick Jason Walls	PMC-Sierra QA Cafe
Chief Editor	Michael Hanrahan	Huawei Technologies

 © The Broadband Forum. All rights reserved

TABLE OF CONTENTS

EXECUTIVE SUMMARY 10

1 PURPOSE AND SCOPE 11

1.1 PURPOSE 11
1.2 SCOPE 12

2 REFERENCES AND TERMINOLOGY 13

2.1 CONVENTIONS 13
2.2 REFERENCES 15
2.3 DEFINITIONS 16
2.4 ABBREVIATIONS 17

3 TECHNICAL REPORT IMPACT 18

3.1 ENERGY EFFICIENCY 18
3.2 IPv6 18
3.3 SECURITY 18
3.4 PRIVACY 18

4 INTRODUCTION TO DEFINING CWMP DATA MODELS USING XML 19

4.1 NAMING CONVENTION FOR DM XML DATA MODEL FILES 19
4.2 NATURAL PROGRESSION OF DM DATA MODELS 20
4.3 SURVEY OF EXISTING XML/XSD FILES 21

5 DM XML DATA MODEL BASICS 24

5.1 ROOT ELEMENT 24
5.2 XML CATALOG 25
5.3 DEFINING ROOT DATA MODELS 27
5.3.1 Key Points – New Root Data Model 27
5.3.2 Basic Walkthrough – Defining a New Root Data Model 27
5.3.3 Key Points – Amended Root Data Model 35
5.3.4 Basic Walkthrough – Defining an Amendment to a Root Data Model 37
5.4 DEFINING SERVICE DATA MODELS 41
5.4.1 Key Points – New Service Data Model 41
5.4.2 Basic Walkthrough – Defining a New Service Data Model 41
5.4.3 Key Points – Amended Service Data Model 46
5.4.4 Basic Walkthrough – Defining an Amendment to a Service Data Model 47
5.5 DEFINING VENDOR-SPECIFIC DATA MODELS 52
5.5.1 Defining a Vendor-Specific Extension to a Standard Data Model 52
5.5.2 Defining a Vendor-Specific Service Data Model 53

6 DM XML DATA MODEL TUTORIALS 54

6.1 BIBLIOGRAPHY 54
6.1.1 Adding Reusable Bibliography Reference Elements 54
6.1.2 Adding Data-Model Specific Bibliography Reference Elements 55
6.1.3 Citing a Bibliographic Reference 56
6.2 NAMED DATA TYPES 58
6.2.1 Define a Basic Named Data Type 58
6.2.2 Define a Derived Named Data Type 59
6.3 IMPORT 61
6.3.1 Import a Named Data Type 62

 © The Broadband Forum. All rights reserved

6.3.2 Import a Data Model ... *62*
6.3.3 Import a Component ... *63*
6.4 MODEL ... 66
6.4.1 Define a New Data Model ... *66*
6.4.2 Extend an Existing Data Model ... *67*
6.4.3 Fixing Errata in an Existing Data Model ... *68*
6.5 OBJECT (DEFINITION) ... 70
6.5.1 Defining a Single-Instance Object ... *70*
6.5.2 Defining a Multi-Instance Object (table) ... *72*
6.5.2.1 Variable-Sized Read-Only Table ... 73
6.5.2.2 Variable-Sized Writable Table (ACS Managed) ... 74
6.5.2.3 Fixed-Sized Table ... 76
6.5.2.4 Unique Key for a Table ... 76
6.5.3 Updating an Existing Object Definition ... *78*
6.6 PARAMETER (DEFINITION) ... 81
6.6.1 Defining a Parameter (The Basics) ... *81*
6.6.1.1 Syntax Using a Built-In Primitive Data Type ... 82
6.6.1.2 Syntax Using a Named Data Type ... 83
6.6.1.3 Refining a Data Type Using Facets ... 84
6.6.1.4 Default Value ... 87
6.6.1.5 Active Notify and Forced Inform ... 88
6.6.2 Number-of-Entries Parameter ... *89*
6.6.3 Hidden-Valued Parameter ... *89*
6.6.4 Command Parameter ... *90*
6.6.5 List-Valued Parameter ... *91*
6.6.6 Reference Parameter ... *93*
6.6.6.1 Path-Reference Parameter ... 94
6.6.6.2 Instance-Reference Parameter ... 97
6.6.6.3 Enumeration-Reference Parameter ... 98
6.6.7 Updating an Existing Parameter Definition ... *100*
6.7 COMPONENT ... 103
6.7.1 Defining and Using a Component ... *103*
6.7.2 Importing and Using a Component ... *106*
6.7.3 Updating an Existing Component ... *106*
6.8 PROFILE ... 110
6.8.1 Defining a New Profile ... *110*
6.8.2 Updating an Existing Profile ... *111*
6.8.3 Defining a New Profile by Extension ... *112*
6.9 DESCRIPTION ... 114
6.9.1 Defining a Description ... *114*
6.9.2 Updating an Existing Description ... *116*
6.9.3 Laying Out Descriptions ... *117*
6.9.3.1 Whitespace Pre-processing ... 117
6.9.3.2 Markup ... 118
6.10 STATUS: DEPRECATE, OBSOLETE, DELETE ... 120

7 DT XML DATA MODEL TUTORIALS ... 122

7.1 BIBLIOGRAPHY ... 123
7.2 IMPORT ... 124
7.2.1 Import a Named Data Type ... *124*
7.2.2 Import a Data Model ... *125*
7.3 MODEL ... 127
7.3.1 Supporting One DM Model ... *127*
7.3.2 Supporting Multiple DM Models ... *128*

 © The Broadband Forum. All rights reserved

7.4 OBJECT 129
7.4.1 Supporting Single-Instance Objects 129
7.4.2 Supporting Multi-Instance Objects 129
7.5 PARAMETER 132
7.5.1 Supporting Parameters (The Basics) 132
7.5.1.1 Supported Syntax and its Data Type 133
7.5.1.2 Supported Data Type Using Facets 134
7.5.1.3 Default Value 135
7.5.2 List-Valued Parameter 136
7.5.3 Path-Reference Parameter 137
7.6 ANNOTATION 139
7.7 FEATURE 140

APPENDIX I – REFERENCE: DATA MODEL XML SCHEMA 141

I.1 DOCUMENT ELEMENT 141
I.2 DESCRIPTION ELEMENT 142
I.2.1 WHITESPACE PRE-PROCESSING 143
I.2.2 MARKUP 144
I.2.3 TEMPLATES 145
I.3 IMPORT ELEMENT 151
I.3.1 IMPORT SUB-ELEMENTS 152
I.4 DATATYPE ELEMENT (DEFINITION) 154
I.5 BIBLIOGRAPHY ELEMENT 155
I.5.1 BIBLIOGRAPHY REFERENCE ELEMENT 156
I.6 COMPONENT ELEMENT (DEFINITION) 157
I.7 COMPONENT ELEMENT (REFERENCE) 158
I.8 MODEL ELEMENT 159
I.9 OBJECT ELEMENT (DEFINITION) 161
I.9.1 OBJECT UNIQUEKEY ELEMENT 165
I.10 PARAMETER ELEMENT (DEFINITION) 166
I.10.1 PARAMETER SYNTAX ELEMENT 168
I.10.2 SYNTAX LIST ELEMENT 170
I.10.3 SYNTAX DATATYPE ELEMENT (REFERENCE) 171
I.10.4 SYNTAX DEFAULT ELEMENT 173
I.11 PROFILE ELEMENT 174
I.11.1 PROFILE OBJECT ELEMENT (REFERENCE) 177
I.11.2 PROFILE PARAMETER ELEMENT (REFERENCE) 178
I.12 BUILT-IN PRIMITIVE DATA TYPE ELEMENTS 180
I.13 DATA TYPE FACETS 182
I.13.1 SIZE ELEMENT 183
I.13.2 INSTANCEREF ELEMENT 184
I.13.3 PATHREF ELEMENT 186
I.13.4 RANGE ELEMENT 189
I.13.5 ENUMERATION ELEMENT 190
I.13.6 ENUMERATIONREF ELEMENT 192
I.13.7 PATTERN ELEMENT 194
I.13.8 UNITS ELEMENT 195

APPENDIX II – REFERENCE: DEVICE TYPE XML SCHEMA 196

II.1 DOCUMENT ELEMENT 196
II.2 ANNOTATION ELEMENT 197
II.3 FEATURE ELEMENT 197

© The Broadband Forum. All rights reserved

II.4 IMPORT ELEMENT 198
II.4.1 IMPORT SUB-ELEMENTS 199
II.5 BIBLIOGRAPHY ELEMENT 200
II.5.1 BIBLIOGRAPHY REFERENCE ELEMENT 201
II.6 MODEL ELEMENT 202
II.7 OBJECT ELEMENT 203
II.8 PARAMETER ELEMENT 205
II.8.1 PARAMETER SYNTAX ELEMENT 206
II.8.2 SYNTAX LIST ELEMENT 207
II.8.3 SYNTAX DATATYPE ELEMENT 209
II.8.4 SYNTAX DEFAULT ELEMENT 210
II.9 BUILT-IN PRIMITIVE DATA TYPE ELEMENTS 211
II.10 DATA TYPE FACETS 214
II.10.1 SIZE ELEMENT 214
II.10.2 PATHREF ELEMENT 215
II.10.3 RANGE ELEMENT 217
II.10.4 ENUMERATION ELEMENT 218
II.10.5 PATTERN ELEMENT 218

APPENDIX III – REFERENCE: DEVICE TYPE FEATURES XML SCHEMA 220

III.1 FEATURE NAMES 220

APPENDIX IV – REFERENCE: DATA MODEL REPORT XML SCHEMA 221

IV.1 DMR ATTRIBUTES 221

APPENDIX V – PROCESSING DATA MODELS, VALIDATING AND REPORTING 226

V.1 XML SCHEMAS AND DATA MODEL DEFINITIONS 226
V.2 XML EDITOR 226
V.3 XML SCHEMA VERIFIER 226
V.4 BBF REPORT TOOL 227
V.4.1 COMMAND LINE VERSION 227
V.4.2 GUI VERSION 229
V.5 PUBLISHED BBF DATA MODEL REPORTS 230

INDEX 231

© The Broadband Forum. All rights reserved

List of Tables

Table 1 – DM `document` attributes 141
Table 2 – DM `document` sub-elements 142
Table 3 – DM `description` attributes 143
Table 4 – Description Markup 144
Table 5 – Description Templates 146
Table 6 – DM `import` attributes 152
Table 7 – DM `import` sub-elements 152
Table 8 – DM `import`'s sub-element attributes 152
Table 9 – DM `dataType` attributes 154
Table 10 – DM `dataType` sub-elements 155
Table 11 – DM `bibliography` sub-elements 156
Table 12 – DM bibliography `reference` attributes 156
Table 13 – DM bibliography `reference` sub-elements 156
Table 14 – DM definition-based `component` attributes 157
Table 15 – DM definition-based `component` sub-elements 158
Table 16 – DM reference-based `component` attributes 159
Table 17 – DM `model` attributes 160
Table 18 – DM `model` sub-elements 161
Table 19 – DM definition-based `object` attributes 161
Table 20 – DM definition-based `object` sub-elements 164
Table 21 – DM `uniqueKey` attributes 165
Table 22 – DM `uniqueKey` sub-elements 165
Table 23 – DM uniqueKey `parameter` attributes 165
Table 24 – DM definition-based `parameter` attributes 166
Table 25 – DM definition-based `parameter` sub-elements 168
Table 26 – DM parameter `syntax` attributes 169
Table 27 – DM parameter `syntax` sub-elements 169
Table 28 – DM syntax `list` attributes 170
Table 29 – DM syntax `list` sub-elements 171
Table 30 – DM syntax `dataType` attributes 172
Table 31 – DM syntax `dataType` sub-elements 172
Table 32 – DM syntax `default` attributes 173
Table 33 – DM syntax `default` sub-elements 174
Table 34 – DM `profile` attributes 175
Table 35 – DM `profile` sub-elements 176
Table 36 – DM profile `object` attributes 177
Table 37 – DM profile `object` sub-elements 178
Table 38 – DM profile `parameter` attributes 179
Table 39 – DM profile `parameter` sub-elements 180
Table 40 – DM primitive data type elements 180
Table 41 – DM data type facet elements 183
Table 42 – DM facet sub-elements 183
Table 43 – DM `size` attributes 183
Table 44 – DM `instanceRef` attributes 184
Table 45 – DM `pathRef` attributes 186
Table 46 – DM `range` attributes 190
Table 47 – DM `enumeration` attributes 191
Table 48 – DM `enumerationRef` attributes 192
Table 49 – DM `pattern` attributes 194
Table 50 – DM `units` attributes 195

© The Broadband Forum. All rights reserved

Table 51 – DT `document` attributes *196*
Table 52 – DT `document` sub-elements *196*
Table 53 – DT `feature` attributes *198*
Table 54 – DT `feature` sub-elements *198*
Table 55 – DT `import` attributes *198*
Table 56 – DT `import` sub-elements *199*
Table 57 – DT `import` sub-element attributes *199*
Table 58 – DT `bibliography` sub-elements *200*
Table 59 – DT `bibliography reference` attributes *201*
Table 60 – DT `bibliography reference` sub-elements *201*
Table 61 – DT `model` attributes *202*
Table 62 – DT `model` sub-elements *202*
Table 63 – DT `object` attributes *203*
Table 64 – DT `object` sub-elements *205*
Table 65 – DT `parameter` attributes *205*
Table 66 – DT `parameter` sub-elements *206*
Table 67 – DT parameter `syntax` sub-elements *206*
Table 68 – DT syntax `list` attributes *208*
Table 69 – DT syntax `list` sub-elements *208*
Table 70 – DT syntax `dataType` attributes *209*
Table 71 – DT syntax `dataType` sub-elements *210*
Table 72 – DT syntax `default` attributes *210*
Table 73 – DT syntax `default` sub-elements *211*
Table 74 – DT primitive data type elements *212*
Table 75 – DT data type facet elements *214*
Table 76 – DT facet sub-elements *214*
Table 77 – DT `size` attributes *215*
Table 78 – DT `pathRef` attributes *215*
Table 79 – DT `range` attributes *217*
Table 80 – DT `enumeration` attributes *218*
Table 81 – DT `pattern` attributes *219*
Table 82 – DMR attributes *221*

© The Broadband Forum. All rights reserved

Executive Summary

TR-154 provides guidance in writing TR-069 XML Data Models that conform to the DM or DT Schemas defined in TR-106 [3]. The XML Data Models produced using these schemas are used for very different purposes; a DM Data Model defines all possible objects and parameters that are available for a particular TR-069 Data Model, while a DT Data Model cites those objects and parameters that a CPE actually supports for a given DM Data Model.

An XML Data Model that conforms to the DM Schema represents a complete definition of objects and parameters that are available for use with TR-069. Both the Broadband Forum and vendor companies define these DM Data Models; the Broadband Forum defines the standard Data Models (e.g. Device:2) while vendor companies can define extensions to the standard Data Models. Vendor companies and SDOs can also define entirely new (vendor-specific) Service Data Models that are unrelated to any of those defined by the Broadband Forum.

An XML Data Model that conforms to the DT Schema describes the supported Data Model for a particular type of device (i.e. it specifies the subset of objects and parameters that a CPE supports for a given DM Data Model). Device vendors specify and publish these DT Data Models. This is external to the Broadband Forum, which has no involvement with supported Data Models.

TR-154 also discusses the Broadband Forum Report Tool, which is used to validate XML Data Models and to generate their HTML and XML Data Model reports. XML Data Models are not easily read by humans; the Report Tool can generate reports that fully describe a Data Model in a concise and readable format.

 © The Broadband Forum. All rights reserved

1 Purpose and Scope

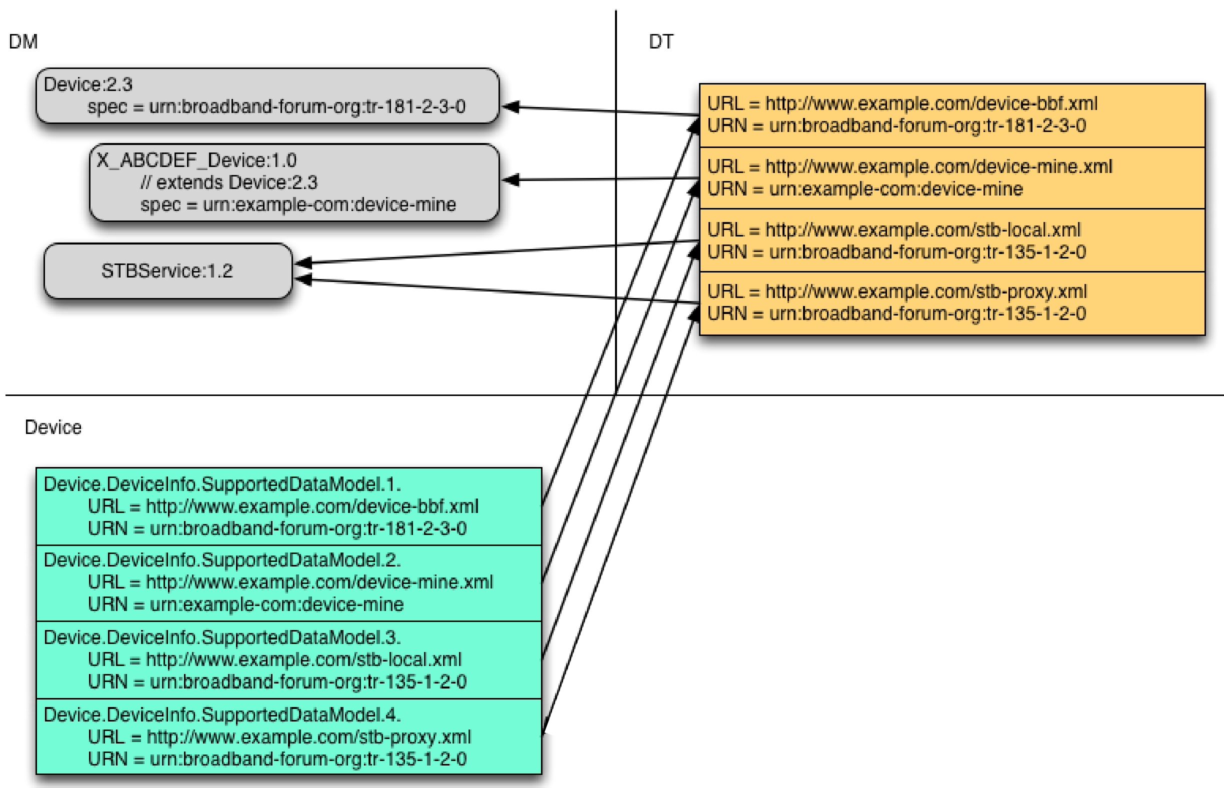
1.1 Purpose

TR-154 provides guidance in writing TR-069 XML Data Models that conform to the DM or DT Schemas defined in TR-106 [3]. The XML Data Models produced using these schemas are used for very different purposes; a DM Data Model defines all possible objects and parameters that are available for a particular TR-069 Data Model, while a DT Data Model cites those objects and parameters that a CPE actually supports for a given DM Data Model.

An XML Data Model that conforms to the DM Schema represents a complete definition of objects and parameters that are available for use with TR-069. Both the Broadband Forum and vendor companies define these DM Data Models; the Broadband Forum defines the standard Data Models (e.g. Device:2) while vendor companies can define extensions to the standard Data Models. Vendor companies and SDOs can also define entirely new (vendor-specific) Service Data Models that are unrelated to any of those defined by the Broadband Forum.

An XML Data Model that conforms to the DT Schema describes the supported Data Model for a particular type of device (i.e. it specifies the subset of objects and parameters that a CPE supports for a given DM Data Model). Device vendors specify and publish these DT Data Models. This is external to the Broadband Forum, which has no involvement with supported Data Models.

The following diagram illustrates the relationships between the DM and DT Instances used by a CPE, and how they tie into the CPE's instantiated Data Model:



 © The Broadband Forum. All rights reserved

1.2 Scope

Section 4 introduces the reader to DM Data Models. It discusses the XML and XSD files that the Broadband Forum has already published, the progression of Data Models over time, as well as "housekeeping" matters such as naming conventions.

Section 5 provides a high-level overview of how DM Data Models are defined. It describes the root element common to all such XML files, the use of XML catalogs in order to more easily indicate where supporting files are located, and basic walkthroughs in defining Root and Service Data Models from the ground up.

Section 6 dives into the details of defining DM Data Models. It discusses the "hows" and "whys" of using the various XML elements and attributes permitted by the DM Schema. Note that the DM Schema itself is specified in Appendix I. While Appendix IV is a reference for the DM Report Schema, an ancillary schema that can be used within DM Data Models in order to provide additional information to reporting tools.

Section 7 dives into the details of defining DT Data Models. It discusses the "hows" and "whys" of using the various XML elements and attributes permitted by the DT Schema. Note that the DT Schema itself is specified in Appendix II. While Appendix III is a reference for the DT Features Schema, an ancillary schema that can be used within DT Data Models in order to indicate supported features.

Appendix V discusses XML tools, one being the BBF Report Tool which is used to validate XML Data Models and to generate their HTML and XML Data Model reports. XML Data Models are not easily read by humans; the Report Tool can generate reports that describe a Data Model in a concise and readable format.

Note – Readers of this Technical Report are expected to be familiar with the Data Model guidelines specified in TR-106 [3]. Furthermore, they should have a basic understanding of XML and XML namespaces [10].

© The Broadband Forum. All rights reserved

2 References and Terminology

TR-154 is solely informative and therefore does not contain any normative text. There are no conventions relating to requirements. TR-154 does provide information on requirements specified in other Broadband Forum documents.

2.1 Conventions

XML

XML examples within TR-154 comply with the XML 1.0 Specification [10]. These examples will be discussed in terms of their elements and attributes. Wikipedia describes elements and attributes based on markup and tags, as follows[1]:

- **Markup and Content** – The characters which make up an XML document are divided into markup and content. Markup and content may be distinguished by the application of simple syntactic rules. All strings which constitute markup either begin with the character "<" and end with a ">", or begin with the character "&" and end with a ";". Strings of characters which are not markup are content.
- **Tag** – A markup construct that begins with "<" and ends with ">". Tags come in three flavors: start-tags, for example `<section>`, end-tags, for example `</section>`, and empty-element tags, for example `<line-break/>`.
- **Element** – A logical document component that either begins with a start-tag and ends with a matching end-tag or consists only of an empty-element tag. The characters between the start- and end-tags, if any, are the element's content, and may contain markup, including other elements, which are called child elements. An example of an element is `<Greeting>Hello, world.</Greeting>`. Another is `<line-break/>`.
- **Attribute** – A markup construct consisting of a name/value pair that exists within a start-tag or empty-element tag. In the example (below) the element `img` has two attributes, `src` and `alt`: `<img src="madonna.jpg" alt='Foligno Madonna, by Raphael'/>`. Another example would be `<step number="3">Connect A to B.</step>` where the name of the attribute is `number` and the value is "3".

Ellipsis

XML examples throughout TR-154 are abbreviated in order to improve readability and to focus on the topic at hand. This is indicated using an ellipsis (…). Whenever an ellipsis is seen within an example, it should be assumed that text has been omitted at that location.

[1] http://en.wikipedia.org/wiki/XML

 © The Broadband Forum. All rights reserved

An example of this is shown below in the XML Path discussion; the ellipsis indicates that, in practice, the model element would include content between its start- and end-tags (e.g. child elements).

Referring to Objects

Throughout this Working Text, object names are often abbreviated in order to improve readability. For example, Device.Ethernet.VLANTermination.{i}. is the full name of a Device:2 object, but might casually be referred to as Ethernet.VLANTermination.{i} or VLANTermination.{i} or VLANTermination, just so long as the abbreviation is unambiguous (with respect to similarly named objects defined elsewhere within the corresponding Data Model).

XML Path

In TR-154, an xml-path shorthand notation is sometimes used to more easily refer to specific elements and attributes within the XML.

For example, the xml-path `document/model` relates to the model elements depicted in the expanded XML shown below. And the xml-path `document/model/@name` relates to the model element's name attribute.

```
<document>
    <model name="Device:2.0">…</model>
</document>
```

This notation is based on the standard XPath notation defined by [13].

 © The Broadband Forum. All rights reserved

2.2 References

The following references are of relevance to this Technical Report. At the time of publication, the editions indicated were valid. All references are subject to revision; users of this Technical Report are therefore encouraged to investigate the possibility of applying the most recent edition of the references listed below.

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

Document		Title	Source	Year
[1]	TR-069 Amendment 4	*CPE WAN Management Protocol*	Broadband Forum	2011
[2]	TR-104	*Provisioning Parameters for VoIP CPE*	Broadband Forum	2005
[3]	TR-106 Amendment 6	*Data Model Template for TR-069-Enabled Devices*	Broadband Forum	2011
[4]	TR-143	*Enabling Network Throughput Performance Tests and Statistical Monitoring*	Broadband Forum	2008
[5]	TR-157 Amendment 5	*Component Objects for CWMP*	Broadband Forum	2011
[6]	TR-181 Issue 1, TR-181 Issue 2 Amendment 2	*Device Data Model for TR-069*	Broadband Forum	2010, 2011
[7]	RFC 2648	*A URN Namespace for IETF Documents*	IETF	1999
[8]	RFC 3986	*Uniform Resource Identifier (URI): Generic Syntax*	IETF	2005
[9]	Organizationally Unique Identifiers (OUIs)		IEEE	
[10]	Extensible Markup Language (XML) 1.0 (Fourth Edition)		W3C	2008
[11]	XML Schema Part 0: Primer Second Edition		W3C	2004
[12]	XML Schema Part 2: Datatypes Second Edition		W3C	2004
[13]	XML Path Language (XPath) Version 1.0		W3C	1999

 © The Broadband Forum. All rights reserved

2.3 Definitions

The following terminology is used throughout this Technical Report.

ACS	Auto-Configuration Server. This is a component in the broadband network responsible for auto-configuration of the *CPE* for advanced services.
Component	A named collection of Objects and/or Parameters and/or Profiles that can be included anywhere within a *Data Model*.
CPE	Customer Premises Equipment; refers to any TR-069-enabled [1] device and therefore covers both Internet Gateway devices and LAN-side end devices.
Data Model	A hierarchical set of Objects and/or Parameters that define the managed objects accessible via TR-069 [1].
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* that is used for defining Data Models for use with CWMP.
DMR Schema	Data Model Report Schema. This is the *XML Schema* that is used for enhancing Data Models with presentation and reporting specific attributes.
DT Instance	Device Type Schema instance document. This is an XML document that conforms to the *DT Schema* and to any additional rules specified in or referenced by the *DT Schema*.
DT Schema	Device Type Schema. This is the *XML Schema* that is used for describing a *Device's Supported Data Model*.
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.
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	Object or Parameter path name. A name that has a hierarchical structure similar to files in a directory, with each level separated by a "." (dot).
Report Tool	Broadband Forum Report Tool. A Perl script that can validate a

 © The Broadband Forum. All rights reserved

	DM or DT Data Model, and generate an associated HTML report file (a human-readable representation of the XML).
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".
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	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. Note that TR-106 [3] further divides this concept into base vs. current supported Data Model.
Template	A Template is a kind of markup. It is encoded within a *DM Instance* description (or *DT Instance* annotation) as text enclosed in double curly braces ({}). Processing tools can replace such Template references with template-dependent text.
XML Schema	An XML schema [11] describes the valid structure of a class of XML documents. An XML schema is specified using the XML Schema Definition (XSD) language.

2.4 Abbreviations

This Technical Report uses the following abbreviations:

BBF	Broadband Forum.
CWMP	CPE WAN Management Protocol [1].
DM	Data Model
DT	Device Type
RFP	Request For Proposal
SDO	Standards Developing Organization
URI	Uniform Resource Identifier [8].
URL	Uniform Resource Locator [8].
XML	Extensible Markup Language [10].
XSD	XML Schema Definition [11].

 © The Broadband Forum. All rights reserved

3 Technical Report Impact

3.1 Energy Efficiency

TR-154 has no impact on energy efficiency.

3.2 IPv6

TR-154 has no impact on IPv6.

3.3 Security

TR-154 has no impact on security.

3.4 Privacy

TR-154 has no impact on privacy.

© The Broadband Forum. All rights reserved

4 Introduction to Defining CWMP Data Models Using XML

A TR-069 Data Model is a hierarchical set of Objects and/or Parameters that define the managed objects accessible via TR-069. All of the TR-069 Data Models are defined using a standard XML Schema.

An XML Data Model that conforms to the DM Schema (as defined in Annex A.3/TR-106 [3]) represents a complete definition of objects and parameters that are available for use with CWMP. Both the Broadband Forum and vendor companies define these DM Data Models; the Broadband Forum defines the standard Data Models (e.g. Device:2) while vendor companies can define extensions to the standard Data Models.

A DM Data Model can consist of one or more XML files that together define a particular version of a Root or Service Data Model. Each version of a Data Model is specified within a primary XML file, but can also import additional information from supporting XML files. Note that all such files conform to the DM Schema, and so each file is a DM Instance.

Note – DM Data Models (and DM Instances) are discussed further in Sections 5 and 6. These should not be confused with DT Data Models (and DT Instances), discussed in Section 7, which CPE vendors instead use to describe their Supported Data Models.

4.1 Naming Convention for DM XML Data Model Files

In the next section we survey some of the published XML files. Each is named according to the Technical Report that defines it. This naming convention is described by the form dd-nnn-i-a-c.xml, where

dd	document type (i.e. tr)
nnn	document number (e.g. 098, 106, 181)
i	document issue number (e.g. 1, 2)
a	document amendment number (e.g. 0, 1, 2)
c	document corrigendum number (e.g. 0, 1, 2)

For example, TR-143 Issue 1 Amendment 0 Corrigendum 2 would declare its Data Model in tr-143-1-0-2.xml. TR-181 Issue 2 Amendment 1 Corrigendum 0 would declare its Data Model in tr-181-2-1-0.xml.

Note – Whenever a Data Model file is referenced without the corrigendum portion of the name, this implies that the latest corrigendum should be used. So, tr-143-1-0.xml is the same as tr-143-1-0-2.xml when corrigendum 2 is the latest revision of TR-143 (Issue 1 Amendment 0).

 © The Broadband Forum. All rights reserved

4.2 Natural Progression of DM Data Models

A new Data Model starts out at version 1.0. Later on, backwards compatible updates (e.g. adding new, or updating existing, objects and parameters) can be made in version 1.1, a minor revision to the Data Model. Additional backwards compatible revisions can follow in this way with Data Model versions 1.2, 1.3, and so on. Generally, each such revision is associated with its own Broadband Forum technical report amendment.

For example, TR-106 (Issue 1 Amendment 0) defined version 1.0 of the Device Data Model (i.e. Device:1.0), declared in tr-106-1-0-0.xml. When backwards compatible changes were needed in the Data Model, then an amendment of TR-106 was created (i.e. TR-106 Issue 1 Amendment 1) in which version 1.1 of the Device Data Model was defined (i.e. Device:1.1), declared in tr-106-1-1-0.xml.

Note – Following the progression of the Device:1 and InternetGatewayDevice:1 Data Models can be challenging since later revisions to these Data Models were defined in completely different technical reports (e.g. different revisions to the Device:1 Data Model can be found in TR-106, TR-143, TR-157, and TR-181 Issue 1). However, going forward, each Root and Service Data Model will always be defined within its corresponding technical report only (and its subsequent amendments).

When changes to a Data Model are limited to correcting errata, then these changes can instead be defined within a corrigendum release (of the relevant amendment). A corrigendum does not result in a new minor revision to the Data Model, but does result in a new physical XML file which will contain the previous definitions (from the file being corrected) as well as the new errata changes. This can be thought of as a re-release of the previous XML file that serves as a replacement. Note that this differs from an amendment release, which does result in a new minor revision to the Data Model, and the corresponding new XML file only contains changes (does not re-specify definitions from earlier revisions).

For example, TR-143 (Issue 1 Amendment 0) defined the Device:1.2 Data Model (declared in tr-143-1-0-0.xml). TR-143 had two subsequent corrigenda releases for Issue 1 Amendment 0. TR-143 (Issue 1 Amendment 0) Corrigendum 2 still defines the Device:1.2 Data Model (albeit with errata corrections and editorial changes added in), however this re-release is declared in tr-143-1-0-2.xml. Since each corrigenda revision is a replacement, if you want to reference the latest corrigendum you do not want to have to know its name, hence the rule that any reference to the file tr-143-1-0.xml should be interpreted as the latest corrigendum defined for TR-143 Issue 1 Amendment 0.

When a Data Model requires non-backwards compatible changes (e.g. moving and redefining objects within the object hierarchy – a re-architecture), then a new major version of the Data Model is started. The new major version is not tied to the previous version. As far as the definitions are concerned, it is like starting over.

For example, TR-181 Issue 2 defines version 2.0 of the Device Data Model (i.e. Device:2.0), declared in tr-181-2-0-0.xml. The Device:2.0 Data Model has no ties to the Device:1.0 Data Model. Backwards compatible updates to the Device:2.0 Data Model will progress in a similar

 © The Broadband Forum. All rights reserved

fashion as was described earlier (e.g. TR-181 Issue 2 Amendment 1 defined the new minor revision Device:2.1 and is declared in tr-181-2-1-0.xml, and so forth).

4.3 Survey of Existing XML/XSD Files

The available XML Schemas and Data Model definitions for the TR-069 suite of documents are listed online at http://www.broadband-forum.org/cwmp.

Below we discuss this family of XML/XSD files. This is not a complete list since existing Data Models and schemas will be revised over time, and new Data Models can also be defined.

Supporting files: These files contain information that can be imported into Data Models. These files are not versioned when updated.

tr-069-1-0-0-types.xml	Contains normative definitions of named (i.e. not built-in) data types that can be used in Data Model definitions. **Note**: This file is not meant to be versioned when updated (the fact that its name includes "1-0-0" is an unfortunate artifact).
tr-069-biblio.xml	Contains a centralized set of bibliographic references for all TR-069 Data Model definitions to use. **Note**: This file is not versioned when updated.

Schema files: Schemas that can be versioned include an x-y suffix in their name, where x is the major version number and y is the minor version number. Note that each revision of a particular schema defines the complete schema (i.e. later revisions do not depend on previous versions for their definitions).

cwmp-datamodel-1-0.xsd cwmp-datamodel-1-1.xsd cwmp-datamodel-1-2.xsd cwmp-datamodel-1-3.xsd cwmp-datamodel-1-4.xsd	TR-069 Data Model Definition Schema (DM Schema). DM Instances define TR-069 Data Models using this schema. **Note**: Each schema revision replaces the version that came before it (the latest revision should be used by new DM Instances). Older DM Schema revisions remain because they may have been statically associated with existing DM Instances.
cwmp-datamodel-report.xsd	TR-069 Data Model Report Schema (DMR Schema). Non-normative definitions that can be used in DM Instances to provide additional information to reporting tools.
cwmp-devicetype-1-0.xsd cwmp-devicetype-1-1.xsd	TR-069 Device Type Schema (DT Schema). DT Instances describe individual devices' support for TR-069 Data Models. **Note**: Each schema revision replaces the version that came before it (the latest revision should be used by new DT Instances). Older DT Schema revisions remain because they may have been statically associated with existing DT Instances.

© The Broadband Forum. All rights reserved

cwmp-devicetype-features.xsd	TR-069 (DT) Device Type Features Schema. Defines device features that can be described in DT Instances.

Root Data Models: Each XML file defines a specific revision to the Device:1, Device:2, or InternetGatewayDevice:1 Data Models. The Data Model version numbering scheme is MODEL:x.y, where x is the major version number and y is the minor version number. Minor (backwards compatible) revisions only contain new content defined since the previous version of the Data Model (i.e. changes and additions). Therefore, the complete definition of a Data Model (e.g. all of Device:1) is the union of all its minor revisions.

InternetGatewayDevice:1	
tr-069-1-0-0.xml	InternetGatewayDevice:1.0 root object definition
tr-098-1-0-0.xml	InternetGatewayDevice:1.1
tr-098-1-1-0.xml	InternetGatewayDevice:1.2
tr-143-1-0-2.xml	InternetGatewayDevice:1.3
tr-098-1-2-0.xml	InternetGatewayDevice:1.4
tr-157-1-0-0.xml	InternetGatewayDevice:1.5
tr-157-1-1-0.xml	InternetGatewayDevice:1.6
tr-157-1-2-0.xml	InternetGatewayDevice:1.7
tr-157-1-3-0.xml	InternetGatewayDevice:1.8
tr-098-1-3-0.xml	InternetGatewayDevice:1.9
Device:1	
tr-106-1-0-0.xml	Device:1.0 root object definition
tr-106-1-1-0.xml	Device:1.1
tr-143-1-0-2.xml	Device:1.2
tr-106-1-2-0.xml	Device:1.2[2]
tr-157-1-0-0.xml	Device:1.3
tr-157-1-1-0.xml	Device:1.4
tr-181-1-0-0.xml	Device:1.5
tr-157-1-2-0.xml	Device:1.6
tr-157-1-3-0.xml	Device:1.7
tr-181-1-1-0.xml	Device:1.8
Device:2	
tr-181-2-0-1.xml	Device:2.0 root object definition
tr-181-2-1-0.xml	Device:2.1
tr-181-2-2-0.xml	Device:2.2
tr-181-2-3-0.xml	Device:2.3

[2] Extends the (like-named/versioned) Device:1.2 Data Model already defined in TR-143 Amendment 1. Basically, this is an update to Device:1.2 without incrementing the model's minor version number. This was done (in this one case) since the updates were purely editorial. This is not the norm.

© The Broadband Forum. All rights reserved

Note – The tr-143 and tr-157 XML files define reusable component objects for CWMP managed devices. Also, going forward, new revisions of the InternetGatewayDevice:1 Data Model will always be defined in a tr-098-1-*a*-*c*.xml file, new revisions of the Device:1 Data Model will always be defined in a tr-181-1-*a*-*c*.xml file, and new revisions of the Device:2 Data Model will always be defined in a tr-181-2-*a*-*c*.xml file (where *a* and *c* are integers corresponding to the amendment and corrigendum of the revision).

Service Data Models: The revision and version numbering scheme used for Root Data Models also applies to Service Data Models.

tr-104-1-0-0.xml	VoiceService:1.0 service object definition
tr-135-1-0-0.xml tr-135-1-1-0.xml	STBService:1.0 service object definition STBService:1.1 service object definition
tr-140-1-0-2.xml tr-140-1-1-0.xml	StorageService:1.0 service object definition StorageService:1.1 service object definition
tr-196-1-0-0.xml tr-196-1-1-0.xml	FAPService:1.0 service object definition FAPService:1.1 service object definition

© The Broadband Forum. All rights reserved

5 DM XML Data Model Basics

5.1 Root Element

Each XML document has exactly one root element. For DM Instances, this is the `document` element (Appendix I.1). It encloses all other elements and therefore is the sole parent element to all the other elements.

The following listing is an example of the `document` element taken from tr-181-2-3-0.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-4"
   xmlns:dmr="urn:broadband-forum-org:cwmp:datamodel-report-0-1"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:broadband-forum-org:cwmp:datamodel-1-4
               http://www.broadband-forum.org/cwmp/cwmp-datamodel-1-4.xsd
            urn:broadband-forum-org:cwmp:datamodel-report-0-1
               http://www.broadband-forum.org/cwmp/cwmp-datamodel-report.xsd"
   spec="urn:broadband-forum-org:tr-181-2-3-0"
   file="tr-181-2-3-0.xml">
  ...
</dm:document>
```

This is a complete (albeit empty) document. Note the following:

- The first line is standard. All XML documents have an `<?xml>` declaration element that looks similar to this in the first line. All Broadband Forum XML documents use UTF-8 encoding.
- `dm:document` is the root element. All XML documents have a single root (top-level) element whose name is determined by the corresponding XML Schema. “dm” means “Data Model” (it references `xmlns:dm`) and is the namespace for the document.
  ***Note**: The name “dm” is the convention used within Broadband Forum XML documents; it really could be anything (e.g. “tns” meaning “This NameSpace” is a common namespace used by some organizations).*
- `xmlns:dm` defines the `dm` in `dm:document`, and references the DM Schema namespace defined by Annex A.3/TR-106 [3] and declared in `cwmp-datamodel-1-4.xsd`.
  ***Note**: The value specified should correspond to the namespace of the latest DM Schema file available at the time of writing.*
- `xmlns:dmr` references the DM Report Schema namespace declared in `cwmp-datamodel-report.xsd`. This attribute is optional; it need only be included if DM Report Schema elements or attributes are used within the document.
- `xmlns:xsi` declares the standard `xsi` (XML Schema Instance) namespace. This value is fixed by the W3C standards organization.
- `xsi:schemaLocation` tells XML editing tools where to find the XML Schemas. Each namespace used within the document (see the `xmlns` attributes above) will have a corresponding entry here. The convention is to specify an absolute path for each XSD file (which is helpful with XML catalogs; see Section 5.2).
  ***Note**: Published Broadband Forum XSD files are available at http://www.broadband-forum.org/cwmp/.*
- `spec` is an attribute defined by DM Schema. Its value indicates the Broadband Forum specification that corresponds to this document. See Appendix A.2.1.1/TR-106 [3] for

 © The Broadband Forum. All rights reserved

guidelines in specifying this value (e.g. the `spec` value for TR-181 Issue 2 Amendment 3 Corrigendum 0 is urn:broadband-forum-org:tr-181-2-3-0).
- `file` is an attribute defined by DM Schema. Its value corresponds to the XML file containing this document. See Appendix A.2.1.1/TR-106 [3] for guidelines in specifying this value (e.g. the `file` value for TR-181 Issue 2 Amendment 3 Corrigendum 0 is tr-181-2-3-0.xml).

Note – A DM Instance document's top-level `file` and `spec` attributes include the corrigendum of the corresponding Broadband Forum specification. This is the only time that corrigenda are included in such attributes. Everywhere else in a document (e.g. on `import` elements where an external document is being referenced), the corrigenda is omitted from `file` and `spec` attributes. The assumption is that the latest corrigendum will always be referenced.

5.2 XML Catalog

An XML catalog[3] provides a mapping from standard URLs to specific local directories on a given machine. For example, a catalog can be used to locate XSD schema files when processing an XML file such as a DM Instance document. This adds a degree of flexibility. Once an XML editor or processing tool has been set up to use a catalog, if the location of the XSD files change, then the catalog can be updated to point to the new location (while the DM Instance documents continue to reference the standard URLs).

As explained in Section 5.1, a DM Instance document's `schemaLocation` attribute will use absolute URLs that have a standard prefix. This is illustrated in the following partial listing.

```
<dm:document
   ...
   xsi:schemaLocation="urn:broadband-forum-org:cwmp:datamodel-1-4
               http://www.broadband-forum.org/cwmp/cwmp-datamodel-1-4.xsd
            urn:broadband-forum-org:cwmp:datamodel-report-0-1
               http://www.broadband-forum.org/cwmp/cwmp-datamodel-report.xsd"
   ...>
```

The published XSD schema files are found on the Broadband Forum website at http://www.broadband-forum.org/cwmp/. When this is not convenient (for example, a new XSD file has not been published yet, or we want to reference the schema files locally rather than over the network) then a catalog can be used to redirect an XML editor or processing tool.

The following listing illustrates a simple XML catalog. Its `systemIdStartString` and `rewritePrefix` attributes indicate that any URL of the form "http://www.broadband-forum.org/cwmp/" will be re-written as "./". Note that the rewrite prefix is relative to the directory that contains the XML catalog. Since this value is "./", it means that the XML catalog should be placed in the same directory as the schema files.

```
<?xml version="1.0"?>
<!DOCTYPE catalog PUBLIC "-//OASIS//DTD Entity Resolution XML Catalog V1.0//EN"
          "http://www.oasis-open.org/committees/entity/release/1.0/catalog.dtd">
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <rewriteSystem systemIdStartString="http://www.broadband-forum.org/cwmp/"
```

[3] See http://en.wikipedia.org/wiki/XML_Catalog for additional information regarding XML catalogs.

 © The Broadband Forum. All rights reserved

```
                 rewritePrefix="./"/>
</catalog>
```

Note that neither the catalog nor the DM Instance documents reference each other. Rather, the catalog is employed by setting up an XML editor (or processing tool) to reference it, whereby URLs within subsequent XML files viewed by the editor (or processing tool) will be resolved accordingly.

 © The Broadband Forum. All rights reserved

5.3 Defining Root Data Models

5.3.1 Key Points – New Root Data Model

The following table outlines the XML elements necessary to define a very basic, initial version of a Root Data Model. For details regarding these elements, see Appendix I.

`document`	Root element. Note that `spec` attribute value corresponds to the Broadband Forum specification that defined the Data Model.
`import`	Used to import content from supporting XML files. In the simple/base case, this is used to import bibliography references and named data type definitions.
`model`	The very first version of a Data Model should be version 1.0. A higher major version is used if the model being defined is a non-backwards compatible revision to an earlier versioned model of the same base name (c.f. Device:1.0 and Device:2.0)
`object`	The `name` attribute will always be used (never the `base` attribute). This is because only new objects are being defined. **Note**: An empty Services object is defined in the initial version of each Root Data Model (just under the top-level object). This serves as the parent object for service objects defined by Service Data Models (e.g. VoiceService)
`parameter`	The `name` attribute will always be used (never the `base` attribute). This is because only new parameters are being defined.
`profile`	The `name` attribute will always be used (never the `base` attribute or the `extends` attribute). This is because only new profiles are being defined.

5.3.2 Basic Walkthrough – Defining a New Root Data Model

This section outlines the basic steps involved in writing a Root Data Model from scratch (i.e. an initial version of a Root Data Model, not an update to an existing Root Data Model). Examples include:

- InternetGatewayDevice:1.0 (declared in tr-069-1-0-0.xml)
- Device:1.0 (declared in tr-106-1-0-0.xml)
- Device:2.0 (declared in tr-181-2-0-0.xml)

This walkthrough is an example of writing the Device:2.0 Data Model.

To begin, the XML Data Model file contains one `document` element (as required for all DM Instance documents). Note that the `spec` attribute references the specification that defined the Data Model, TR-181 Issue 2 in this case. Also, the DM Schema v1.2 specified by the `dm` and `schemaLocation` attributes was the latest version available when the Device:2.0 Data Model was written[4].

[4] Note that the required `document/@file` attribute is not specified. This is because it is part of DM Schema v1.4.

© The Broadband Forum. All rights reserved

```
<?xml version="1.0" encoding="UTF-8"?>
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:broadband-forum-org:cwmp:datamodel-1-2 cwmp-datamodel-1-2.xsd"
   spec="urn:broadband-forum-org:tr-181-2-0-0">
  ...
</dm:document>
```

Add a document description and model element. The model name (i.e. Device:2.0) is assigned a minor version number of 0 since this is the initial version of the Data Model.

```
  <description>Device:2.0 Data Model.</description>

  <model name="Device:2.0">
    ...
  </model>
```

Putting it all together we have:

```
<?xml version="1.0" encoding="UTF-8"?>
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:broadband-forum-org:cwmp:datamodel-1-2 cwmp-datamodel-1-2.xsd"
   spec="urn:broadband-forum-org:tr-181-2-0-0">

  <description>Device:2 Data Model.</description>

  <model name="Device:2.0">
    ...
  </model>

</dm:document>
```

Now add the top-level object within the model. The four attributes specified on the object (`name`, `access`, `minEntries`, and `maxEntries`) are mandatory for objects.

```
    <object name="Device." access="readOnly" minEntries="1" maxEntries="1">
      <description>The top-level object for a Device.</description>
      ...
    </object>
```

Add a parameter within the Device object. Notice that we have also included a `description` element immediately under the `object` and `parameter` elements. Descriptions can be included within almost any element. Note that `description` when used is always the first child element; e.g. the object description comes before the object parameters or any other elements within the object). The two attributes specified on the parameter (`name` and `access`) are mandatory for parameters.

```
    <object name="Device." access="readOnly" minEntries="1" maxEntries="1">
      <description>The top-level object for a Device.</description>

      <parameter name="InterfaceStackNumberOfEntries" access="readOnly">
        <description>{{numentries}}</description>
        <syntax>
          <unsignedInt/>
        </syntax>
      </parameter>

    </object>
```

Note the use of the `{{numentries}}` Template in the above description. In this case, it is equivalent to text such as “The number of entries in the InterfaceStack table”. For details see Section 6.6.2.

© The Broadband Forum. All rights reserved

A Services (single-instance) object will be defined in each Root Data Model. Note that we do not define objects or parameters under the Services object; these are placed here by a CPE at run-time from objects and parameters defined in the Service Data Models (e.g. TR-104 defines the VoiceService:1.0 service objects).

```
    <object name="Device.Services." access="readOnly" minEntries="1" maxEntries="1">
      <description>This object contains general services information.</description>
    </object>
```

Putting it all together we have:

```
<?xml version="1.0" encoding="UTF-8"?>
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:broadband-forum-org:cwmp:datamodel-1-2 cwmp-datamodel-1-2.xsd"
   spec="urn:broadband-forum-org:tr-181-2-0-0">

  <description>Device:2 Data Model.</description>

  <model name="Device:2.0">

    <object name="Device." access="readOnly" minEntries="1" maxEntries="1">
      <description>The top-level object for a Device.</description>

      <parameter name="InterfaceStackNumberOfEntries" access="readOnly">
        <description>{{numentries}}</description>
        <syntax>
          <unsignedInt/>
        </syntax>
      </parameter>

    </object>

    <object name="Device.Services." access="readOnly" minEntries="1" maxEntries="1">
      <description>This object contains general services information.</description>
    </object>

  </model>

</dm:document>
```

Note that all object elements are defined immediately under the model element (i.e. object elements sharing a common hierarchy are not physically nested). However, an object's position within the object hierarchy is clear since its name is in fact a Path Name.

Now continue to build up the object hierarchy. Define the Ethernet object below the Device object. This hierarchy is indicated using a Path Name value within the Ethernet object's `name` attribute (i.e. "Device.Ethernet." indicates that this Ethernet object sits within the Device object).

```
    <object name="Device.Ethernet." access="readOnly" minEntries="1" maxEntries="1">
      <description>Ethernet object. This object models...</description>
      ...
    </object>
```

Add the Ethernet.Interface table (a multi-instance object) and its associated InterfaceNumberOfEntries parameter. Points of interest with this table include the presence of the `numEntriesParameter` attribute and the value of the `maxEntries` attribute. Note that the `numEntriesParameter` attribute (and its corresponding parameter definition) is always specified for a table that has a variable number of entries.

```
    <object name="Device.Ethernet." access="readOnly" minEntries="1" maxEntries="1">
      ...
```

 © The Broadband Forum. All rights reserved

```
      <parameter name="InterfaceNumberOfEntries" access="readOnly">
        <description>{{numentries}}</description>
        <syntax>
          <unsignedInt/>
        </syntax>
      </parameter>
    </object>

    <object name="Device.Ethernet.Interface.{i}." access="readOnly"
            numEntriesParameter="InterfaceNumberOfEntries" minEntries="0"
            maxEntries="unbounded">
      <description>Ethernet interface table. This table models...</description>
      ...
    </object>
```

Add the Ethernet.Link table. Points of interest include the presence of the `numEntriesParameter` attribute and the `enableParameter` attribute. The `enableParameter` attribute indicates the name of the parameter that will enable and disable a table entry; this attribute is required on writable tables when new entries must be configured with a unique key value prior to being enabled. Note that read-only tables can also define an enable parameter, but there is no need to declare the corresponding `enableParameter` attribute.

```
    <object name="Device.Ethernet." access="readOnly" minEntries="1" maxEntries="1">
      ...
      <parameter name="LinkNumberOfEntries" access="readOnly">
        <description>{{numentries}}</description>
        <syntax>
          <unsignedInt/>
        </syntax>
      </parameter>
    </object>

    <object name="Device.Ethernet.Link.{i}." access="readWrite"
            numEntriesParameter="LinkNumberOfEntries" enableParameter="Enable"
            minEntries="0" maxEntries="unbounded">
      <description>Ethernet link layer table. This table models...</description>

      <parameter name="Enable" access="readWrite">
        <description>Enables or disables the link.</description>
        <syntax>
          <boolean/>
        </syntax>
      </parameter>

      ...
    </object>
```

Putting it all together we have:

© The Broadband Forum. All rights reserved

```
<?xml version="1.0" encoding="UTF-8"?>
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:broadband-forum-org:cwmp:datamodel-1-2 cwmp-datamodel-1-2.xsd"
   spec="urn:broadband-forum-org:tr-181-2-0-0">
  <description>Device:2 Data Model.</description>

  <model name="Device:2.0">

    <object name="Device." access="readOnly" minEntries="1" maxEntries="1">
      <description>The top-level object for a Device.</description>
      <parameter name="InterfaceStackNumberOfEntries" access="readOnly">
        <description>{{numentries}}</description>
        <syntax>
          <unsignedInt/>
        </syntax>
      </parameter>
    </object>

    <object name="Device.Services." access="readOnly" minEntries="1" maxEntries="1">
      <description>This object contains general services information.</description>
    </object>

    <object name="Device.Ethernet." access="readOnly" minEntries="1" maxEntries="1">
      <description>Ethernet object. This object models...</description>

      <parameter name="InterfaceNumberOfEntries" access="readOnly">
        <description>{{numentries}}</description>
        <syntax>
          <unsignedInt/>
        </syntax>
      </parameter>

      <parameter name=" LinkNumberOfEntries " access="readOnly">
        <description>{{numentries}}</description>
        <syntax>
          <unsignedInt/>
        </syntax>
      </parameter>
    </object>

    <object name="Device.Ethernet.Interface.{i}." access="readOnly"
            numEntriesParameter="InterfaceNumberOfEntries" minEntries="0" maxEntries="unbounded">
      <description>Ethernet interface table. This table models...</description>
      ...
    </object>

    <object name="Device.Ethernet.Link.{i}." access="readWrite" numEntriesParameter="LinkNumberOfEntries"
            enableParameter="Enable" minEntries="0" maxEntries="unbounded">
      <description>Ethernet link layer table. This table models...</description>

      <parameter name="Enable" access="readWrite">
        <description>Enables or disables the link.</description>
        <syntax>
          <boolean/>
        </syntax>
      </parameter>
      ...
    </object>

  </model>

</dm:document>
```

Add unique keys to the Ethernet.Interface and Ethernet.Link tables. Note that a table's unique key references a corresponding parameter definition within the table (e.g. Name parameter). See Section 6.5.2.4 re: functional vs. non-functional keys.

 © The Broadband Forum. All rights reserved

```
    <object name="Device.Ethernet.Interface.{i}." access="readOnly"
            numEntriesParameter="InterfaceNumberOfEntries" minEntries="0" maxEntries="unbounded">
      ...
      <uniqueKey functional="false">
        <parameter ref="Name"/>
      </uniqueKey>

      <parameter name="Name" access="readOnly">
        <description>The textual name of the interface as assigned by the CPE.</description>
        <syntax>
          <string>
            <size maxLength="64"/>
          </string>
        </syntax>
      </parameter>
      ...
    </object>

    <object name="Device.Ethernet.Link.{i}." access="readWrite"
            numEntriesParameter="LinkNumberOfEntries" enableParameter="Enable" minEntries="0"
            maxEntries="unbounded">
      ...
      <uniqueKey functional="false">
        <parameter ref="Name"/>
      </uniqueKey>

      <parameter name="Name" access="readOnly">
        <description>The textual name of the link as assigned by the CPE.</description>
        <syntax>
          <string>
            <size maxLength="64"/>
          </string>
        </syntax>
      </parameter>
      ...
    </object>
```

Import the bibliography XML file (allows bibliography references to be cited from within `description` elements throughout the document). Also import the data types XML file. Both are supporting XML files that can be used by Data Models. These `import` elements go below the `document` element and above the `model` element. Specify the specific data types to be imported (i.e. only need to import those data types that will actually be used within the document).

```
  <import file="tr-069-biblio.xml" spec="urn:broadband-forum-org:tr-069-biblio"/>

  <import file="tr-106-1-0-types.xml" spec="urn:broadband-forum-org:tr-106-1-0">
    <dataType name="IPPrefix"/>
    <dataType name="IPAddress"/>
    <dataType name="IPv4Address"/>
    <dataType name="MACAddress"/>
  </import>
```

Now update object and parameter definitions to make use of the imported bibliography and data type files. Points of interest include the `{{bibref|RFC2863}}` description Template which references an entry from the imported bibliography XML, and the `dataType` element which references a named type from the imported data types XML.

```
    <object name="Device.Ethernet.Link.{i}." ...>
      ...
      <parameter name="Enable" access="readWrite">
        <description>
          Enables or disables the link.
          This parameter is based on ''ifAdminStatus'' from {{bibref|RFC2863}}.
        </description>
```

 © The Broadband Forum. All rights reserved

```
      ...
    </parameter>

    <parameter name="MACAddress" access="readOnly">
      <description>The MAC address used for packets sent via this interface.</description>
      <syntax>
        <dataType ref="MACAddress"/>
      </syntax>
    </parameter>
  </object>
```

Add a couple of profiles. Note that objects and parameters within a `profile` element do not define new objects/parameters; rather, they reference existing objects and parameters defined earlier within the Data Model (via the `ref` attribute). Also note the `requirement` attribute, which is used to declare additional requirements on these objects and parameters.

Every Data Model should probably have a Baseline profile. Other profiles can be defined based on use cases. Often, profiles are used to describe requirements in RFPs.

```
  <profile name="Baseline:1">
    <object ref="Device." requirement="present">
      <parameter ref="InterfaceStackNumberOfEntries" requirement="readOnly"/>
    </object>
    ...
  </profile>

  <profile name="EthernetLink:1">
    <object ref="Device.Ethernet." requirement="present">
      <parameter ref="LinkNumberOfEntries" requirement="readOnly"/>
    </object>
    <object ref="Device.Ethernet.Link.{i}." requirement="createDelete">
      <parameter ref="Enable" requirement="readWrite"/>
      <parameter ref="Name" requirement="readOnly"/>
      <parameter ref="MACAddress" requirement="readOnly"/>
      ...
    </object>
  </profile>
```

Putting it all together we have:

© The Broadband Forum. All rights reserved

```
<?xml version="1.0" encoding="UTF-8"?>
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:broadband-forum-org:cwmp:datamodel-1-2 cwmp-datamodel-1-2.xsd”
   spec="urn:broadband-forum-org:tr-181-2-0-0">
  <description>Device:2 Data Model.</description>

  <import file="tr-069-biblio.xml" spec="urn:broadband-forum-org:tr-069-biblio"/>

  <import file="tr-106-1-0-types.xml" spec="urn:broadband-forum-org:tr-106-1-0">
    <dataType name="IPPrefix"/>
    <dataType name="IPAddress"/>
    <dataType name="IPv4Address"/>
    <dataType name="MACAddress"/>
  </import>

  <model name="Device:2.0">

    <object name="Device." access="readOnly" minEntries="1" maxEntries="1">
      <description>The top-level object for a Device.</description>
      <parameter name="InterfaceStackNumberOfEntries" access="readOnly">
        <description>{{numentries}}</description>
        <syntax>
          <unsignedInt/>
        </syntax>
      </parameter>
    </object>

    <object name="Device.Services." access="readOnly" minEntries="1" maxEntries="1">
      <description>This object contains general services information.</description>
    </object>

    <object name="Device.Ethernet." access="readOnly" minEntries="1" maxEntries="1">
      <description>Ethernet object. This object models...</description>
      <parameter name="InterfaceNumberOfEntries" access="readOnly">
        <description>{{numentries}}</description>
        <syntax>
          <unsignedInt/>
        </syntax>
      </parameter>
      <parameter name=" LinkNumberOfEntries " access="readOnly">
        <description>{{numentries}}</description>
        <syntax>
          <unsignedInt/>
        </syntax>
      </parameter>
    </object>

    <object name="Device.Ethernet.Interface.{i}." access="readOnly"
            numEntriesParameter="InterfaceNumberOfEntries" minEntries="0" maxEntries="unbounded">
      <description>Ethernet interface table. This table models...</description>

      <uniqueKey functional="false">
        <parameter ref="Name"/>
      </uniqueKey>

      <parameter name="Name" access="readOnly">
        <description>The textual name of the interface as assigned by the CPE.</description>
        <syntax>
          <string>
            <size maxLength="64"/>
          </string>
        </syntax>
      </parameter>
      ...
    </object>

    <object name="Device.Ethernet.Link.{i}." access="readWrite" numEntriesParameter="LinkNumberOfEntries"
            enableParameter="Enable" minEntries="0" maxEntries="unbounded">
      <description>Ethernet link layer table. This table models...</description>
```

 © The Broadband Forum. All rights reserved

```
      <uniqueKey functional="false">
        <parameter ref="Name"/>
      </uniqueKey>

      <parameter name="Enable" access="readWrite">
        <description>
          Enables or disables the link.
          This parameter is based on ''ifAdminStatus'' from {{bibref|RFC2863}}.
        </description>
        <syntax>
          <boolean/>
        </syntax>
      </parameter>

      <parameter name="Name" access="readOnly">
        <description>The textual name of the link as assigned by the CPE.</description>
        <syntax>
          <string>
            <size maxLength="64"/>
          </string>
        </syntax>
      </parameter>

      <parameter name="MACAddress" access="readOnly">
        <description>The MAC address used for packets sent via this interface.</description>
        <syntax>
          <dataType ref="MACAddress"/>
        </syntax>
      </parameter>
      ...
    </object>

    <profile name="Baseline:1">
      <object ref="Device." requirement="present">
        <parameter ref="InterfaceStackNumberOfEntries" requirement="readOnly"/>
      </object>
      ...
    </profile>

    <profile name="EthernetLink:1">
      <object ref="Device.Ethernet." requirement="present">
        <parameter ref="LinkNumberOfEntries" requirement="readOnly"/>
      </object>
      <object ref="Device.Ethernet.Link.{i}." requirement="createDelete">
        <parameter ref="Enable" requirement="readWrite"/>
        <parameter ref="Name" requirement="readOnly"/>
        <parameter ref="MACAddress" requirement="readOnly"/>
        ...
      </object>
    </profile>

  </model>

</dm:document>
```

5.3.3 Key Points – Amended Root Data Model

The following table outlines the basic XML elements necessary to define a minor revision (amendment) to a Root Data Model. This applies whether it is the first or the nth revision. For details regarding these elements, see Appendix I.

`document`	Root element. Note that `spec` attribute value corresponds to the Broadband Forum specification that defined the Data Model.
`import`	Used to import the previous version of the Data Model (in this way, elements defined in previous versions of the Data Model are made "visible" in the

© The Broadband Forum. All rights reserved

	current version of the Data Model). Also used to import content from supporting XML files. In the simple/base case, this is used to import named data type definitions. No need to import the bibliography XML file, as it will carry over as part of importing the previous version of the Data Model.
`model`	Both the `name` and `base` attribute will be used together to create a new version of the Data Model that is being extended. Such a Data Model revision should increment the Data Model's minor version number by one (the major version should remain unchanged, as the intent here is to indicate a backwards compatible update to the previous version of the Data Model).
`object`	The `name` attribute will be used to define new objects. The `base` attribute will be used to update existing objects that were defined in an earlier version of the Data Model.
`parameter`	The `name` attribute will be used to define new parameters. The `base` attribute will be used to update existing parameters that were defined in an earlier version of the Data Model.
`profile`	The `name` attribute will be used to define new profiles. Both the `name` and `base` attribute will be used together to create a new version of existing profiles that were defined in an earlier version of the Data Model.

In a minor revision to a Data Model, only items that are updated or added (e.g. objects, parameters, profiles) will appear in the document. Items that were defined in a previous minor version of the Data Model, and that are not changing in this revision, will not be included in this new revision's XML file.

Note that items that are being updated do not need to be completely re-specified (only the changes need to be specified). This applies to an item's contained elements and optional attributes, where omitted elements and attributes signify "no change" from how they were defined in a previous revision[5]. For example, a table defintion (multi-instance `object`) that is only being updated to include a new parameter will not re-specify its optional `numEntriesParameter` attribute, nor will it re-define its `uniqueKey` element, nor will it specify existing parameters that have not changed. Such elements and optional attributes would only be included if they need to be updated in some way.

[5] When an item is updated, an omitted optional attribute (e.g. object/@status) is treated as "no change" from its previous definition. This is not the case when an item is first defined; where, an omitted optional attribute implies some default value (e.g. object/@status="current" by default).

© The Broadband Forum. All rights reserved

5.3.4 Basic Walkthrough – Defining an Amendment to a Root Data Model

This section outlines the basic steps involved in extending an existing Root Data Model (i.e. creating a minor revision). This applies whether it is the first or n^{th} revision to the Data Model. Examples include:

- InternetGatewayDevice:1.4 (declared in tr-098-1-2-0.xml)
- Device:1.5 (declared in tr-181-1-0-0.xml)
- Device:2.2 (declared in tr-181-2-2-0.xml)

The main differences in writing an amendment, rather than an initial, Root Data Model are:

- The previous version of the Data Model needs to be imported
- The model/@base attribute references the previous version of the Data Model that was imported.
- For object, parameter, and profile definitions being updated, their object/@base, parameter/@base, and profile/@base attributes are used to reference the previous definition.
- For descriptions being updated, the description/@action attribute indicates how the specified description will be applied to its previous definition (i.e. prefix, append, replace).

This walkthrough is an example of writing the Device:2.2 Data Model.

To begin, the XML Data Model file contains one `document` element (as required for all DM Instance documents). Note that the `spec` attribute references the specification that defined the Data Model, TR-181 Issue 2 Amendment 2 in this case. Also, the DM Schema v1.3 specified by the `dm` and `schemaLocation` attributes was the latest version available when the Device:2.2 Data Model was written[6].

```
<?xml version="1.0" encoding="UTF-8"?>
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:broadband-forum-org:cwmp:datamodel-1-3 cwmp-datamodel-1-3.xsd"
   spec="urn:broadband-forum-org:tr-181-2-2-0">

  <description>Device:2.2 Data Model.</description>

  ...
</dm:document>
```

Import previous version of the Data Model (i.e. import Device:2.1 from TR-181 Issue 2 Amendment 1), and then define the new Device:2.2 version of the Data Model which will be based on (built on top of) the existing definitions imported from Device:2.1.

The `model` element uses its `base` attribute to indicate which Data Model (and version) is being extended, and its `name` attribute to indicate the name (and version) of the updated Data Model. Note that the Data Model's minor version is incremented by one.

```
  <import file="tr-181-2-1.xml" spec="urn:broadband-forum-org:tr-181-2-1">
    <model name="Device:2.1"/>
  </import>
```

[6] Note that the required `document/@file` attribute is not specified. This is because it is part of DM Schema v1.4.

© The Broadband Forum. All rights reserved

```
  <model name="Device:2.2" base="Device:2.1">
    ...
  </model>
```

Import named data types that will be referenced in this revision of the Data Model.

```
  <import file="tr-106-1-0-types.xml" spec="urn:broadband-forum-org:tr-106-1-0">
    <dataType name="IPv6Address"/>
    <dataType name="IPv6Prefix"/>
    <dataType name="IPv4Address"/>
    <dataType name="MACAddress"/>
  </import>
```

Putting it all together we have:

```
<?xml version="1.0" encoding="UTF-8"?>
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:broadband-forum-org:cwmp:datamodel-1-3 cwmp-datamodel-1-3.xsd"
    spec="urn:broadband-forum-org:tr-181-2-2-0">

  <description>Device:2.2 Data Model.</description>

  <import file="tr-106-1-0-types.xml" spec="urn:broadband-forum-org:tr-106-1-0">
    <dataType name="IPv6Address"/>
    <dataType name="IPv6Prefix"/>
    <dataType name="IPv4Address"/>
    <dataType name="MACAddress"/>
  </import>

  <import file="tr-181-2-1.xml" spec="urn:broadband-forum-org:tr-181-2-1">
    <model name="Device:2.1"/>
  </import>

  <model name="Device:2.2" base="Device:2.1">
    ...
  </model>
  ...
</dm:document>
```

Define an update to the existing ManagementServer object and to its existing ConnectionRequestURL parameter. Notice the use of the `base` attribute to reference existing objects and parameters. Update the parameter description by appending additional text to the end of its existing description (the `action` attribute set to “append” stipulates this behavior; cf. the “prefix” and “replace” action values).

Note that we are re-specifying the object and parameter attributes (e.g. `access`, `minEntries`, `forcedInform`, etc.) in order to mirror the original definition of these elements, and because in some cases these are required attributes and it would be a schema violation to omit them. We do not specify the object description (since we are not making changes to the existing text), and we do not specify the parameter syntax (since nothing in its syntax changed).

```
   <object base="Device." access="readOnly" minEntries="1" maxEntries="1">
   </object>

   <object base="Device.ManagementServer." access="readOnly" minEntries="1" maxEntries="1">
     <parameter base="ConnectionRequestURL" access="readOnly" forcedInform="true"
                activeNotify="forceDefaultEnabled">
        <description action="append">
          Note: If the ''host'' portion of the URL is a literal IPv6 address then it MUST be...
        </description>
     </parameter>
   </object>
```


© The Broadband Forum. All rights reserved

Define a new parameter under an existing object defined in an earlier version of the Data Model. Again, reference the existing object using the `base` attribute rather than `name` (i.e. declare rather than define). Note that defining new parameters in an amendment is done in the usual way using the `name` attribute. In this example we are defining a parameter that is a list of enumerated items.

```
    <object base="Device.DNS." access="readOnly" minEntries="1" maxEntries="1">
      <parameter name="SupportedRecordTypes" access="readOnly">
        <description>The DNS record types that are supported by the device.</description>
        <syntax>
          <list/>
          <string>
            <enumeration value="A"/>
            <enumeration value="AAAA"/>
            <enumeration value="SRV"/>
            <enumeration value="PTR"/>
          </string>
        </syntax>
      </parameter>
    </object>
```

Update an existing profile. The `profile` element uses the `name` attribute to define the new profile, but in this case the `base` attribute is also used to reference the existing profile that is being extended. The name of the profile is suffixed with its version number. The updated profile version number is one greater than the previous version. Objects and parameters declared within the profile are simply references (see `ref` attribute) to the actual object and parameters that were defined earlier in the Data Model.

```
    <profile name="Baseline:2" base="Baseline:1">
      <object ref="Device.DNS." requirement="present">
        <parameter ref="SupportedRecordTypes" requirement="readOnly"/>
      </object>
    </profile>
```

Putting it all together we have:

 © The Broadband Forum. All rights reserved

```
<?xml version="1.0" encoding="UTF-8"?>
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:broadband-forum-org:cwmp:datamodel-1-3 cwmp-datamodel-1-3.xsd”
    spec="urn:broadband-forum-org:tr-181-2-2-0">

  <description>Device:2.2 Data Model.</description>

  <import file="tr-106-1-0-types.xml" spec="urn:broadband-forum-org:tr-106-1-0">
    <dataType name="IPv6Address"/>
    <dataType name="IPv6Prefix"/>
    <dataType name="IPv4Address"/>
    <dataType name="MACAddress"/>
  </import>

  <import file="tr-181-2-1.xml" spec="urn:broadband-forum-org:tr-181-2-1">
    <model name="Device:2.1"/>
  </import>

  <model name="Device:2.2" base=”Device:2.1”>
    ...
    <object base="Device.ManagementServer." access="readOnly" minEntries="1" maxEntries="1">
      <parameter base="ConnectionRequestURL" access="readOnly" forcedInform="true"
                 activeNotify="forceDefaultEnabled">
        <description action="append">
          Note: If the ''host'' portion of the URL is a literal IPv6 address then it MUST be ...
        </description>
      </parameter>
    </object>

    <object base="Device.DNS." access="readOnly" minEntries="1" maxEntries="1">
      <parameter name="SupportedRecordTypes" access="readOnly">
        <description>The DNS record types that are supported by the device.</description>
        <syntax>
          <list/>
          <string>
            <enumeration value="A"/>
            <enumeration value="AAAA"/>
            <enumeration value="SRV"/>
            <enumeration value="PTR"/>
          </string>
        </syntax>
      </parameter>
    </object>
    ...

    <profile name="Baseline:2" base="Baseline:1">
      <object ref="Device.DNS." requirement="present">
        <parameter ref="SupportedRecordTypes" requirement="readOnly"/>
      </object>
    </profile>

    ...
  </model>
  ...
</dm:document>
```


© The Broadband Forum. All rights reserved

5.4 Defining Service Data Models

5.4.1 Key Points – New Service Data Model

The following table outlines the XML elements necessary to define a very basic initial version of a Service Data Model. For details regarding these elements, see Appendix I.

`document`	Root element. Note that `spec` attribute value corresponds to the Broadband Forum specification that defined the Data Model.
`import`	Used to import content from supporting XML files. In the simple/base case, this is used to import bibliography references and named data type definitions.
`model`	The very first version of a Data Model should be version 1.0. A higher major version is used if the model being defined is a non-backwards compatible revision to an earlier versioned model of the same base name (c.f. Device:1.0 and Device:2.0). The `isService` attribute will be present and have value “true”.
`object`	The `name` attribute will always be used (never the `base` attribute). This is because only new objects are being defined. **Note**: All Service Data Models have a top-level table (e.g. VoiceService.{i}.), where the table name coincides with the name of the service model. This table is the root of the service model hierarchy. This differs from the Root Data Models, which have either the Device or InternetGatewayDevice singleton object at the root of their hierarchy.
`parameter`	The `name` attribute will always be used (never the `base` attribute). This is because only new parameters are being defined. **Note**: All Service Data Models have a top-level number-of-entries parameter (e.g. VoiceServiceNumberOfEntries) that is associated with the top-level service table.
`profile`	The `name` attribute will always be used (never the `base` attribute or the `extends` attribute). This is because only new profiles are being defined.

5.4.2 Basic Walkthrough – Defining a New Service Data Model

This section outlines the basic steps involved in writing a Service Data Model from scratch (i.e. an initial version of a Service Data Model, not an update to an existing Service Data Model). Examples include:

- VoiceService:1.0 (declared in tr-104-1-0-0.xml)
- STBService:1.0 (declared in tr-135-1-0-0.xml)
- FAPService:1.0 (declared in tr-196-1-0-0.xml)

In many ways, writing a Service Data Model is quite similar to writing a Root Data Model. The main differences with a Service Data Model are:

 © The Broadband Forum. All rights reserved

- The `model` element includes the `isService` attribute, set to true.
- There is a top-level multi-instance object, whose name coincides with the name of the service model. This is the top of this service model hierarchy.
- There is a top-level number-of-entries parameter that is associated with the top-level multi-instance service object.
- There is no Device.Services (or InternetGatewayDevice.Services) object, which is only defined in a Root Data Model.

This walkthrough is an example of writing the VoiceService:1.0 Data Model.

To begin, the XML Data Model file contains one `document` element (as required for all DM Instance documents). Note that the `spec` attribute references the specification that defined the Data Model, TR-104 Issue 1 in this case. Also, the DM Schema v1.1 specified by the `dm` and `schemaLocation` attributes was the latest version available when the VoiceService:1.0 Data Model was written[7].

```
<?xml version="1.0" encoding="UTF-8"?>
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:broadband-forum-org:cwmp:datamodel-1-0 cwmp-datamodel-1-0.xsd"
   spec="urn:broadband-forum-org:tr-104-1-0-0">
  ...
</dm:document>
```

Add a document description and model element. The `model` element uses the `isService` attribute to indicate that this is a Service Data Model; the model `name` is assigned a minor version number of 0 since this is the initial version of the Data Model.

```
  <description>VoiceService:1 Data Model.</description>

  <model name="VoiceService:1.0" isService="true">
    ...
  </model>
```

Import the bibliography XML file (allows bibliography references to be cited from within `description` elements throughout the document). Also import the data types XML file. Specify the specific data types to be imported (i.e. only need to import those data types that will actually be used within the document). Both of these `import` elements go below the `document` and above the `model` element.

```
  <import file="tr-069-biblio.xml" spec="urn:broadband-forum-org:tr-069-biblio"/>

  <import file="tr-106-1-0-types.xml" spec="urn:broadband-forum-org:tr-106-1-0">
    <dataType name="IPAddress"/>
  </import>
```

Putting it all together

[7] Note that the required `document/@file` attribute is not specified. This is because it is part of DM Schema v1.4.

 © The Broadband Forum. All rights reserved

```
<?xml version="1.0" encoding="UTF-8"?>
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:broadband-forum-org:cwmp:datamodel-1-0 cwmp-datamodel-1-0.xsd"
   spec="urn:broadband-forum-org:tr-104-1-0-0">

  <description>VoiceService:1 Data Model.</description>

  <import file="tr-069-biblio.xml" spec="urn:broadband-forum-org:tr-069-biblio"/>

  <import file="tr-106-1-0-types.xml" spec="urn:broadband-forum-org:tr-106-1-0">
    <dataType name="IPAddress"/>
    <dataType name="MACAddress"/>
  </import>

  <model name="VoiceService:1.0" isService="true">
    ...
  </model>

</dm:document>
```

Add the top-level VoiceService table (a multi-instance object) and its associated top-level VoiceServiceNumberOfEntries parameter (i.e. these are at the top/root of the object hierarchy). Note that all Service Data Models have such a top-level table and associated number-of-entries parameter, where the table name coincides with the name of the Service Data Model. Other points of interest include the presence of the `numEntriesParameter` attribute.

```
    <parameter name="VoiceServiceNumberOfEntries" access="readOnly">
      <description>{{numentries}}</description>
      <syntax>
        <unsignedInt/>
      </syntax>
    </parameter>

    <object name="VoiceService.{i}." access="readOnly" minEntries="0" maxEntries="unbounded"
            numEntriesParameter="VoiceServiceNumberOfEntries">
      <description>The top-level object for VoIP CPE.</description>

      <parameter name="VoiceProfileNumberOfEntries" access="readOnly">
        <description>{{numentries}}</description>
        <syntax>
          <unsignedInt/>
        </syntax>
      </parameter>

    </object>
```

Now continue to build up the object hierarchy. Since this is the initial version of the Data Model, all objects and parameters are defined using the `name` attribute. And each object indicates its relative position within the hierarchy using the path-dot-notation within its `name` (e.g. an object with name "VoiceServices.{i}.Capabilities." indicates that this Capabilities object sits within the VoiceService.{i} object).

Note the FarEndIPAddress parameter below; it is defined using a `dataType` element that references the IPAddress data type. This is possible here because we imported the data types XML file earlier (which is where the IPAddress data type is actually defined).

```
    <object name="VoiceService.{i}.Capabilities." access="readOnly" minEntries="1"
            maxEntries="1">
      <description>The overall capabilities of the VoIP CPE.</description>

      <parameter name="MaxProfileCount" access="readOnly" activeNotify="canDeny">
        <description>Maximum total number of distinct voice profiles supported.</description>
```


© The Broadband Forum. All rights reserved

```
      <syntax>
        <unsignedInt/>
      </syntax>
    </parameter>
    ...
  </object>

  <object name="VoiceService.{i}.VoiceProfile.{i}." ...>
    ...
  </object>

  <object name="VoiceService.{i}.VoiceProfile.{i}.Line.{i}." ...>
    ...
  </object>

  <object name="VoiceService.{i}.VoiceProfile.{i}.Line.{i}.Session.{i}." access="readOnly"
          minEntries="0" maxEntries="unbounded" >
    <description>Information on each active ...</description>
    ...
    <parameter name="FarEndIPAddress" access="readOnly" activeNotify="canDeny">
      <description>The IP address of far end VoIP device.</description>
      <syntax>
        <dataType ref="IPAddress"/>
      </syntax>
    </parameter>
  </object>
```

Add a profile. Note that objects and parameters within a `profile` element do not define new objects/parameters; rather, they reference existing objects and parameters defined earlier within the Data Model (via the `ref` attribute). Also note the `requirement` attribute, which is used to declare additional requirements on these objects and parameters.

```
  <profile name="Endpoint:1">
    <object ref="VoiceService.{i}." requirement="present">
      <parameter ref="VoiceProfileNumberOfEntries" requirement="readOnly"/>
    </object>

    <object ref="VoiceService.{i}.Capabilities." requirement="present">
      <parameter ref="MaxProfileCount" requirement="readOnly"/>
      ...
    </object>

    <object ref="VoiceService.{i}.VoiceProfile.{i}." requirement="createDelete">
      ...
    </object>

    <object ref="VoiceService.{i}.VoiceProfile.{i}.Line.{i}." requirement="createDelete">
      ...
    </object>

    <object ref="VoiceService.{i}.VoiceProfile.{i}.Line.{i}.Session.{i}."
            requirement="present">
      ...
      <parameter ref="FarEndIPAddress" requirement="readOnly"/>
    </object>
    ...
  </profile>
```

Putting it all together

 © The Broadband Forum. All rights reserved

```
<?xml version="1.0" encoding="UTF-8"?>
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:broadband-forum-org:cwmp:datamodel-1-0 cwmp-datamodel-1-0.xsd"
   spec="urn:broadband-forum-org:tr-104-1-0-0">

  <description>VoiceService:1 Data Model.</description>

  <import file="tr-069-biblio.xml" spec="urn:broadband-forum-org:tr-069-biblio"/>

  <import file="tr-106-1-0-types.xml" spec="urn:broadband-forum-org:tr-106-1-0">
    <dataType name="IPAddress"/>
    <dataType name="MACAddress"/>
  </import>

  <model name="VoiceService:1.0" isService="true">
    ...
    <parameter name="VoiceServiceNumberOfEntries" access="readOnly">
      <description>{{numentries}}</description>
      <syntax>
        <unsignedInt/>
      </syntax>
    </parameter>

    <object name="VoiceService.{i}." access="readOnly" minEntries="0" maxEntries="unbounded"
            numEntriesParameter="VoiceServiceNumberOfEntries">
      <description>The top-level object for VoIP CPE.</description>
      <parameter name="VoiceProfileNumberOfEntries" access="readOnly">
        <description>{{numentries}}</description>
        <syntax>
          <unsignedInt/>
        </syntax>
      </parameter>
    </object>

    <object name="VoiceService.{i}.Capabilities." access="readOnly" minEntries="1" maxEntries="1">
      <description>The overall capabilities of the VoIP CPE.</description>
      <parameter name="MaxProfileCount" access="readOnly" activeNotify="canDeny">
        <description>Maximum total number of distinct voice profiles supported.</description>
        <syntax>
          <unsignedInt/>
        </syntax>
      </parameter>
      ...
    </object>

    <object name="VoiceService.{i}.VoiceProfile.{i}." ...>
      ...
    </object>

    <object name="VoiceService.{i}.VoiceProfile.{i}.Line.{i}." ...>
      ...
    </object>

    <object name="VoiceService.{i}.VoiceProfile.{i}.Line.{i}.Session.{i}." access="readOnly" minEntries="0"
            maxEntries="unbounded" >
      <description>Information on each active ...</description>
      ...
      <parameter name="FarEndIPAddress" access="readOnly" activeNotify="canDeny">
        <description>The IP address of far end VoIP device.</description>
        <syntax>
          <dataType ref="IPAddress"/>
        </syntax>
      </parameter>
    </object>
    ...

    <profile name="Endpoint:1">
      <object ref="VoiceService.{i}." requirement="present">
        <parameter ref="VoiceProfileNumberOfEntries" requirement="readOnly"/>
      </object>
```

 © The Broadband Forum. All rights reserved

```
      <object ref="VoiceService.{i}.Capabilities." requirement="present">
        <parameter ref="MaxProfileCount" requirement="readOnly"/>
        ...
      </object>
      <object ref="VoiceService.{i}.VoiceProfile.{i}." requirement="createDelete">
        ...
      </object>
      <object ref="VoiceService.{i}.VoiceProfile.{i}.Line.{i}." requirement="createDelete">
        ...
      </object>
      <object ref="VoiceService.{i}.VoiceProfile.{i}.Line.{i}.Session.{i}." requirement="present">
        ...
        <parameter ref="FarEndIPAddress" requirement="readOnly"/>
      </object>
      ...
    </profile>
    ...
  </model>

</dm:document>
```

5.4.3 Key Points – Amended Service Data Model

The following table outlines the basic XML elements necessary to define a minor revision (amendment) to a Service Data Model. This applies whether it is the first or the nth revision. For details regarding these elements, see Appendix I.

`document`	Root element. Note that `spec` attribute value corresponds to the Broadband Forum specification that defined the Data Model.
`import`	Used to import the previous version of the Data Model (in this way, elements defined in previous versions of the Data Model are made "visible" in the current version of the Data Model). Also used to import content from supporting XML files. In the simple/base case, this is used to import named data type definitions. No need to import the bibliography XML file, as it will carry over as part of importing the previous version of the Data Model.
`model`	Both the `name` and `base` attribute will be used together to create a new version of the Data Model that is being extended. Such a Data Model revision should increment the Data Model's minor version number by one (the major version should remain unchanged, as the intent here is to indicate a backwards compatible update to the previous version of the Data Model).
`object`	The `name` attribute will be used to define new objects. The `base` attribute will be used to update existing objects that were defined in an earlier version of the Data Model. **Note**: All Service Data Models have a top-level table (e.g. VoiceService.{i}.), where the table name coincides with the name of the service model (this will have been defined in the initial version of the Data Model). This table is the root of the service model hierarchy.
`parameter`	The `name` attribute will be used to define new parameters. The `base`

 © The Broadband Forum. All rights reserved

	attribute will be used to update existing parameters that were defined in an earlier version of the Data Model. **Note**: All Service Data Models have a top-level number-of-entries parameter (e.g. VoiceServiceNumberOfEntries) that is associated with the top-level service table (this will have been defined in the initial version of the Data Model).
`profile`	The `name` attribute will be used to define new profiles. Both the `name` and `base` attribute will be used together to create a new version of existing profiles that were defined in an earlier version of the Data Model.

In a minor revision to a Data Model, only items that are updated or added (e.g. objects, parameters, profiles) will appear in the document. Items that were defined in a previous minor version of the Data Model, and that are not changing in this revision, will not be included in this new revision's XML file.

Note that items that are being updated do not need to be completely re-specified (only the changes need to be specified). An item's omitted elements and optional attributes signify "no change" from how they were defined in a previous revision. This concept applies to both Root and Service Data Model amendments (see Section 5.3.3 for further explanation).

5.4.4 Basic Walkthrough – Defining an Amendment to a Service Data Model

This section outlines the basic steps involved in extending an existing Service Data Model (i.e. creating a minor revision). This applies whether it is the first or nth revision to the Data Model. Examples include:

- VoiceService:1.1 (declared in tr-104-1-1-0.xml)
- STBService:1.1 (declared in tr-135-1-1-0.xml)
- FAPService:1.1 (declared in tr-196-1-1-0.xml)

The main differences in writing an amendment, rather than an initial, Service Data Model are:

- The previous version of the Data Model needs to be imported
- The model/@base attribute references the previous version of the Data Model that was imported.
- For object, parameter, and profile definitions being updated, their object/@base, parameter/@base, and profile/@base attributes are used to reference the previous definition.
- For descriptions being updated, the description/@action attribute indicates how the specified description will be applied to its previous definition (i.e. prefix, append, replace)

This walkthrough is an example of writing the VoiceService:1.1 Data Model.

To begin, the XML Data Model file contains one `document` element (as required for all DM Instance documents). Note that the `spec` attribute references the specification that defined the Data Model, TR-104 Issue 1 Amendment 1 in this case. Also, the DM Schema v1.3 specified by

 © The Broadband Forum. All rights reserved

the `dm` and `schemaLocation` attributes was the latest version available when the VoiceService:1.1 Data Model was written[8].

```
This section outlines the basic steps involved in <?xml version="1.0" encoding="UTF-8"?>
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:broadband-forum-org:cwmp:datamodel-1-3 cwmp-datamodel-1-3.xsd"
   spec="urn:broadband-forum-org:tr-104-1-1-0">

  <description>VoiceService:1.1 Data Model.</description>

  ...
</dm:document>
```

Import previous version of the Data Model (i.e. import VoiceService:1.0 from TR-104 Issue 1), and then define the new VoiceService:1.1 version of the Data Model which will be based on (built on top of) the existing definitions imported from VoiceService:1.0.

The `model` element uses its `base` attribute to indicate which Data Model (and version) is being extended, and its `name` attribute to indicate the name (and version) of the updated Data Model. Note that the Data Model's minor version is incremented by one. Note that the `model` element uses the `isService` attribute to indicate that this is a Service Data Model.

```
  <import file="tr-104-1-0.xml" spec="urn:broadband-forum-org:tr-104-1-0">
    <model name="VoiceService:1.0"/>
  </import>

  <model name="VoiceService:1.1" base="VoiceService:1.0" isService="true">
    ...
  </model>
```

Import named data types that will be referenced in this revision of the Data Model.

```
  <import file="tr-106-1-0-types.xml" spec="urn:broadband-forum-org:tr-106-1-0">
    <dataType name="IPAddress"/>
    <dataType name="MACAddress"/>
  </import>
```

Putting it all together

[8] Note that the required `document/@file` attribute is not specified. This is because it is part of DM Schema v1.4.

 © The Broadband Forum. All rights reserved

```
<?xml version="1.0" encoding="UTF-8"?>
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:broadband-forum-org:cwmp:datamodel-1-3 cwmp-datamodel-1-3.xsd"
   spec="urn:broadband-forum-org:tr-104-1-1-0">

  <description>VoiceService:1.1 Data Model.</description>

  <import file="tr-106-1-0-types.xml" spec="urn:broadband-forum-org:tr-106-1-0">
    <dataType name="IPAddress"/>
  </import>

  <import file="tr-104-1-0.xml" spec="urn:broadband-forum-org:tr-104-1-0">
    <model name="VoiceService:1.0"/>
  </import>

  <model name="VoiceService:1.1" base="VoiceService:1.0" isService="true">
    ...
  </model>

  ...
</dm:document>
```

Define an update to the existing SignalingProtocols parameter under the VoiceServices.{i}.-Capabilities object. Notice the use of the `base` attribute to reference existing objects and parameters. Update the parameter description by appending additional text to the end of its existing description (the `action` attribute set to “append” stipulates this behavior; cf. the “prefix” and “replace” action values).

Note that we are re-specifying the object and parameter attributes (e.g. `access`, `minEntries`, `activeNotify`, etc.) in order to mirror the original definition of these elements, and because in some cases these are required attributes and it would be a schema violation to omit them. Notice that we have not re-specified the object and parameter descriptions (this is because we are not making changes to the existing description text). However, do note that the parameter’s full set of pattern facets are re-specified even though we are just adding two new patterns; this is because a data type facet must be fully specified when changed (see Section 6.6.1.3).

Define the new CallForwarding parameter under the VoiceServices.{i}.Capabilities object. Notice the use of the `name` attribute to define a new parameter. Defining new objects and parameters, or updating existing ones, is done here in the same manner as in any other Data Model.

```
    <object base="VoiceService.{i}." access="readOnly" minEntries="0" maxEntries="unbounded">
    </object>

    <object base="VoiceService.{i}.Capabilities." access="readOnly" minEntries="1"
            maxEntries="1">

      <parameter base="SignalingProtocols" access="readOnly" activeNotify="canDeny">
        <description action="append">Added enumeration values: "POTS", and "ISDN".</description>
        <syntax>
          <string>
            <pattern value="SIP"/>
            <pattern value="MGCP"/>
            <pattern value="MGCP-NCS"/>
            <pattern value="H\.248"/>
            <pattern value="H\.323"/>
            <pattern value="POTS"/>
            <pattern value="ISDN"/>
            <pattern value="SIP/\d+\.\d+"/>
            <pattern value="MGCP/\d+\.\d+"/>
```

 © The Broadband Forum. All rights reserved

```
          <pattern value="MGCP-NCS/\d+\.\d+"/>
          <pattern value="H\.248/\d+\.\d+"/>
          <pattern value="H\.323/\d+\.\d+"/>
          <pattern value="X_.+"/>
        </string>
      </syntax>
    </parameter>

    <parameter name="CallForwarding" access="readOnly">
      <description>Support for call forwarding.</description>
      <syntax>
        <boolean/>
      </syntax>
    </parameter>
    ...
  </object>
```

This amendment does not define any profiles.

Putting it all together

 © The Broadband Forum. All rights reserved

```
<?xml version="1.0" encoding="UTF-8"?>
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:broadband-forum-org:cwmp:datamodel-1-3 cwmp-datamodel-1-3.xsd"
   spec="urn:broadband-forum-org:tr-104-1-1-0">

  <description>VoiceService:1.1 Data Model.</description>

  <import file="tr-106-1-0-types.xml" spec="urn:broadband-forum-org:tr-106-1-0">
    <dataType name="IPAddress"/>
  </import>

  <import file="tr-104-1-0.xml" spec="urn:broadband-forum-org:tr-104-1-0">
    <model name="VoiceService:1.0"/>
  </import>

  <model name="VoiceService:1.1" base="VoiceService:1.0" isService="true">

    <object base="VoiceService.{i}." access="readOnly" minEntries="0" maxEntries="unbounded">
    </object>

    <object base="VoiceService.{i}.Capabilities." access="readOnly" minEntries="1" maxEntries="1">
      <parameter base="SignalingProtocols" access="readOnly" activeNotify="canDeny">
        <description action="append">Added enumeration values: "POTS", and "ISDN".</description>
        <syntax>
          <string>
            <pattern value="SIP"/>
            <pattern value="MGCP"/>
            <pattern value="MGCP-NCS"/>
            <pattern value="H\.248"/>
            <pattern value="H\.323"/>
            <pattern value="POTS"/>
            <pattern value="ISDN"/>
            <pattern value="SIP/\d+\.\d+"/>
            <pattern value="MGCP/\d+\.\d+"/>
            <pattern value="MGCP-NCS/\d+\.\d+"/>
            <pattern value="H\.248/\d+\.\d+"/>
            <pattern value="H\.323/\d+\.\d+"/>
            <pattern value="X_.+"/>
          </string>
        </syntax>
      </parameter>
      <parameter name="CallForwarding" access="readOnly">
        <description>Support for call forwarding.</description>
        <syntax>
          <boolean/>
        </syntax>
      </parameter>
      ...
    </object>

    ...
  </model>

  ...
</dm:document>
```


© The Broadband Forum. All rights reserved

5.5 Defining Vendor-Specific Data Models

5.5.1 Defining a Vendor-Specific Extension to a Standard Data Model

Vendors can extend Broadband Forum Data Models in order to define their own items (e.g. vendor-specific objects, parameters, enumerations, patterns, profiles). Both Root and Service Data Models can be extended in this way; such extensions are defined in the same way Broadband Forum amendments extend Data Models. However, the name assigned to vendor-specific items will begin with X_<VENDOR>_, where <VENDOR> must be as defined in Section 3.3/TR-106 [3].

A vendor-specific XML Data Model contains one `document` element. The document's `file` and `spec` attributes will have a vendor-specific doc number. The Broadband Forum model to be extended is imported and used to define the vendor-specific model.

The following listing is an example of a vendor-specific model named X_ACDC73_Device:2.1 (where the vendor's designation is ACDC73). Note that the Device:2.1 model is imported into the vendor-specific document, and is used to define the vendor-specific model based on Device:2.1. The model is extended in order to add a vendor-specific object and parameter.

```
<?xml version="1.0" encoding="UTF-8"?>
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-4"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:broadband-forum-org:cwmp:datamodel-1-4 cwmp-datamodel-1-4.xsd"
    file="acdc73_tr-181-2-1-0.xml" spec="urn:acdc73:tr-181-2-1-0">

  <description>Device:2.1 Data Model with vendor extensions.</description>
  ...

  <import file="tr-181-2-1.xml" spec="urn:broadband-forum-org:tr-181-2-1">
    <model name="Device:2.1"/>
  </import>

  <model name="X_ACDC73_Device:2.1" base="Device:2.1">

    <object name="Device.ManagementServer.X_ACDC73_ExampleObject." ... >
      <description>...</description>

      <parameter name="ExampleParameter" ...>
        <description>...</description>
        <syntax>
          <unsignedInt/>
        </syntax>
      </parameter>
      ...
    </object>
    ...

  </model>
</dm:document>
```

In the above example, ExampleParameter is a vendor-specific parameter but its name does not include the vendor prefix. This is because its parent object is vendor-specific. Vendor-specific items (e.g. objects, parameters, enumerations) will not be named with a vendor prefix when their parent object or parameter is vendor-specific. In other words, this vendor-prefix naming rule only applies to the boundary between the Broadband Forum items and vendor-specific items.

© The Broadband Forum. All rights reserved

5.5.2 Defining a Vendor-Specific Service Data Model

Vendors can also define entirely new Service Data Models independent of the Broadband Forum models. In this case, the entire model is vendor-specific.

Defining such a vendor-specific Service Data Model is done in the same way as defining a Broadband Forum Service Data Model (see Section 5.4). The only difference is that the top-level items will be named with an X_<VENDOR>_ prefix, where <VENDOR> must be as defined in Section 3.3/TR-106 [3]. The following items will be named with a vendor prefix:

- The `model`.
- The top-level number-of-entries `parameter`.
- The top-level multi-instance service `object`.

Note that items defined within the top-level multi-instance `object` (e.g. objects, parameters, enumerations) will not be named with a vendor prefix.

The following listing is an example of such a vendor-specific Service Data Model (where the vendor's designation is ACDC73). Note that document's `file` and `spec` attributes have a vendor-specific doc number.

```
<?xml version="1.0" encoding="UTF-8"?>
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-4"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:broadband-forum-org:cwmp:datamodel-1-4 cwmp-datamodel-1-4.xsd"
    file="acdc73_example-1-0-0.xml" spec="urn:acdc73:example-1-0-0">

  <description>Vendor ExampleService:1.0 Data Model.</description>
  ...

  <model name="X_ACDC73_ExampleService:1.0" isService="true">

    <parameter name="X_ACDC73_ExampleServiceNumberOfEntries" access="readOnly">
      <description>{{numentries}}</description>
      <syntax>
        <unsignedInt/>
      </syntax>
    </parameter>

    <object name="X_ACDC73_ExampleService.{i}." access="readOnly" minEntries="0" maxEntries="unbounded"
            numEntriesParameter="X_ACDC73_ExampleServiceNumberOfEntries">
      <description>...</description>

      <parameter name="ExampleParameter" ...>
        <description>...</description>
        <syntax>
          <unsignedInt/>
        </syntax>
      </parameter>
      ...

    </object>

  </model>
</dm:document>
```


© The Broadband Forum. All rights reserved

6 DM XML Data Model Tutorials

This section provides guidance in defining DM Instance documents (called DM Instances for short). These documents are XML files that comply with the DM Schema (Appendix I). They are defined by the Broadband Forum in order to specify standard Data Models and by CPE vendors in order to specify vendor-specific Data Models. Such Data Models represent objects and parameters available to a range of device types (i.e. not specific to a particular device type).

A DM Data Model can be specified using multiple DM Instance documents. This will include a primary DM Instance document (that defines the latest revision of a given model), previous DM Instance documents (that define the previous revisions of the model if present), as well as supporting DM Instance documents (that define elements such as components, data types, and bibliography). See Sections 4 and 5 for further introductory materials regarding DM Instances and Data Models.

Note – DM Instances (and DM Data Models) should not be confused with DT Instances (and DT Data Models), discussed in Section 7, which instead describe a CPE vendor's Supported Data Model for a particular device type.

6.1 Bibliography

Bibliography references are defined using the `bibliography/reference` element (I.5.1). There are two ways to define bibliography references:

- within the central bibliography file, or
- within a given Data Model file

Regardless of where a bibliography reference is defined, it can be cited from within descriptions in the same manner.

Note – Defining references within the central bibliography file means that they can be reused across Data Models.

Note – The DM Schema indicates that bibliographies are defined under a `document` element, which means that they can be defined within any XML file. This might be useful when a draft document is in development. However, the convention is to limit bibliography definitions to the central bibliography file for all published documents.

6.1.1 Adding Reusable Bibliography Reference Elements

The centralized, global bibliography is defined in tr-069-biblio.xml. This file lists both Broadband Forum and external specifications that are cited in the various XML Data Models. It most cases, it is best to use this central file to define bibliography reference elements.

The following listing illustrates the layout and key elements found in tr-069-biblio.xml (see I.5 for details on the bibliography element). Two bibliography references are defined in this example, RFC 4122 and TR-181i2a1.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3" ...
```

 © The Broadband Forum. All rights reserved

```
           spec="urn:broadband-forum-org:tr-069-biblio">
  <bibliography>
    ...
    <reference id="RFC4122">
      <name>RFC 4122</name>
      <title>A Universally Unique Identifier (UUID) URN Namespace</title>
      <organization>IETF</organization>
      <category>RFC</category>
      <date>2005</date>
      <hyperlink>http://tools.ietf.org/rfc/rfc4122.txt</hyperlink>
    </reference>
    ...
    <reference id="TR-181i2a1">
      <name>TR-181 Issue 2 Amendment 1</name>
      <title>Device Data Model for TR-069</title>
      <organization>Broadband Forum</organization>
      <category>TR</category>
      <date>2010</date>
    </reference>
    ...
  <bibliography>
</dm:document>
```

To add a new bibliography reference to tr-069-biblio.xml, simply insert a `bibliography/reference` element and its related sub-elements. At a minimum, the reference needs to include an `id`, `name`, `title`, and should also include `organization`, `category`, `date`, and `hyperlink` where applicable. Broadband Forum bibliography references do not need to include a hyperlink to the published specification, since this information is well-known and can be generated by the Report Tool.

The `id` is intended to uniquely identify a bibliography reference across all DM Instance documents (i.e. it should be globally unique). Appendix A.2.4/TR-106 [3] specifies a set of rules used to determine the standard id value for specifications published by various SDOs. These rules can be summarized through example[9]:

- "TR-181" refers to BBF technical report TR-181 Issue 1 (Amendment 0)
- "TR-106a2" refers to BBF technical report TR-106 Issue 1 Amendment 2
- "TR-181i2a1" refers to BBF technical report TR-181 Issue 2 Amendment 1
- "RFC4078" refers to IETF RFC 4078
- "802.1D-2004" refers to IEEE 802.1D-2004
- "G.998.3" refers to ITU-T G.998.3

This id is used by XML Data Models in order to cite specific references from within `description` elements.

Note – A Data Model file will need to import tr-069-biblio.xml in order to use its bibliography.

6.1.2 Adding Data-Model Specific Bibliography Reference Elements

Defining bibliography references from within a given Data Model file is accomplished in much the same way as is done using tr-069-biblio.xml (see 6.1.1). Although extending tr-069-biblio.xml is preferred, if the reference is highly specific to the Data Model in question then it

[9] An id value will not contain space characters. An id value for a Broadband Forum technical report will not include the corrigendum; the latest corrigendum is always assumed.

 © The Broadband Forum. All rights reserved

can be defined directly within the Data Model file. But doing this means that this bibliography reference cannot be reused by other (unrelated) Data Models.

The following listing illustrates the definition a bibliography reference from within a Data Model file (see I.5 for details on the bibliography element).

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3" ...
             spec="urn:broadband-forum-org:tr-135-1-1-0">
  ...
  <import file="tr-069-biblio.xml" spec="urn:broadband-forum-org:tr-069-biblio"/>
  ...
  <bibliography>
    ...
    <reference id="RFC4078">
      <name>RFC 4078</name>
      <title>The TV-Anytime Content Reference Identifier (CRID)</title>
      <organization>IETF</organization>
      <category>RFC</category>
      <hyperlink>http://www.ietf.org/rfc/rfc4078.txt</hyperlink>
    </reference>
    ...
  <bibliography>

  <model...> ... </model>
</dm:document>
```

A new bibliography reference is defined within the `document` using a `bibliography/-reference` element. This is exactly the same as in the tr-069-biblio.xml file.

The `bibliography` element comes after the `description` and `import` elements (if present), and before other top-level elements such as `model` and `component`.

Note – In the above listing, importing tr-069-biblio.xml is optional. Doing so is only necessary if the Data Model cites references defined in that file.

Note – Be careful not to duplicate references already defined within the central tr-069-biblio.xml file. The Report Tool includes a `warnbibref` setting to warn against such duplications (see V.4.1).

6.1.3 Citing a Bibliographic Reference

Bibliography references can be cited from within any `description` element in the Data Model; for example, from within object descriptions, parameter descriptions, enumeration descriptions, etc. This is done by using a `{{bibref}}` Template (see I.2.3).

In order to cite a bibliographic reference, the reference definition must be "visible" from within the local file making the citation. This is accomplished by either importing the file that defines the reference or by defining the reference directly within the local Data Model file.

In the following Data Model listing, the global bibliography file tr-069-biblio.xml is imported in order to gain access to its defined reference elements. Then the EnabledOptions parameter description cites TR-069 Table 48 using the notation `{{bibref|TR-069|Table 48}}`. The second part of the Template is the id of the bibliographic reference being cited, and the (optional) third part of the Template is free-form text indicating a specific area of the document being cited.

 © The Broadband Forum. All rights reserved

In this case it is `Table 48`, however, this value will vary depending on the area of the document being cited (e.g. Section, Appendix, Annex, Table, Figure, etc.).

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-0" ...
             spec="urn:broadband-forum-org:tr-069-1-0-0">
  ...
  <import file="tr-069-biblio.xml" spec="urn:broadband-forum-org:tr-069-biblio"/>
  ...
  <model...>
    <object ...>
      <parameter name="EnabledOptions" access="readOnly">
        <description>... as described in {{bibref|TR-069|Table 48}}.</description>
      </parameter>
      ...
    </object>
    ...
</model>
</dm:document>
```

In the following Data Model listing, bibliography reference RFC 4078 is defined directly within the local file. Then the ContentReferenceId parameter description cites RFC 4078 using the notation `{{bibref|RFC4078}}`. The `RFC4078` portion is the id of the bibliographic reference being cited. In this case the reference is visible (available to be used within bibref Templates) because it is defined directly within the local file.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3" ...
             spec="urn:broadband-forum-org:tr-135-1-1-0">
  ...
  <bibliography>
    <reference id="RFC4078">
      <name>RFC 4078</name>
      <title>The TV-Anytime Content Reference Identifier (CRID)</title>
      <organization>IETF</organization>
      <category>RFC</category>
      <hyperlink>http://www.ietf.org/rfc/rfc4078.txt</hyperlink>
    </reference>
    ...
  <bibliography>

  <model...>
    <object ...>
      ...
      <parameter name="ContentReferenceId" access="readOnly">
        <description>Unique Content Item reference as defined in {{bibref|RFC4078}}</description>
        ...
      </parameter>
    </object>
    ...
</model>
</dm:document>
```

 © The Broadband Forum. All rights reserved

6.2 Named Data Types

A named data type is a custom data type that is usually defined within the central tr-106-1-0-0-types.xml file using the top-level `dataType` element (I.4). This should not be confused with the dataType element that can appear within parameter definitions, which is just a data type reference (a reference to a named data type defined elsewhere) rather than a data type definition.

A named data type should be defined when there is a set of data requirements that can apply to a range of parameters. For example, an IP address or a MAC address. By defining and using a named data type, it guarantees that these data requirements will be applied consistently.

A named data type is defined in terms of one of the built-in primitive data types (e.g. unsignedInt), or defined in terms of another named data type from which it inherits part of its definition.

The two varieties of named data types can be characterized as: basic and derived. A derived type will inherit from another named data type (either basic or derived). A basic type inherits nothing and is defined using one of the built-in primitive data types.

The ultimate purpose in defining a named data type is in specifying a parameter's type (i.e. within a `parameter/syntax/dataType` element). Using a named data type is a consistent way to associate a set of data requirements with a parameter. See Section 6.6.1.2 for details.

Note – The DM Schema indicates that named data types are defined under a `document` element, which means that data types can be defined within any XML file. This might be useful when a draft document is in development. However, the convention is to limit data type definitions to the central tr-106-1-0-0-types.xml file for all published documents.

Note – The tr-106-1-0-0-types.xml file name does not change when it is updated.

Note – A Data Model file will need to import tr-106-1-0-0-types.xml in order to use its data types. See Section 6.3.1 for details.

6.2.1 Define a Basic Named Data Type

A basic named data type is defined using one (and only one) of the built-in primitive types. These built-in types are (see I.12):

- base64
- boolean
- dateTime
- hexBinary
- int
- long
- string
- unsignedInt
- unsignedLong

 © The Broadband Forum. All rights reserved

To add a new data type to tr-106-1-0-0-types.xml, simply insert a `dataType` element and its related sub-elements (I.4). At a minimum, the data type needs to include a `name`, a `description`, and exactly one built-in type.

The following listing illustrates the definition of the StatsCounter32 data type (defined using the primitive `unsignedInt` type).

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3" ...
             spec="urn:broadband-forum-org:tr-106-1-0-0">
  ...
  <dataType name="StatsCounter32">
    <description>A 32-bit statistics parameter, e.g. a byte counter.</description>
    <unsignedInt/>
  </dataType>
  ...
</dm:document>
```

The description is optional but recommended, and can be used to specify additional data requirements. The name must be unique as it is used to identify the data type across Data Models. The name must begin with an upper-case letter, in order to avoid confusion with built-in data types.

The following listing illustrates the definition of the MACAddress data type (defined using the primitive `string` type).

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3" ...
             spec="urn:broadband-forum-org:tr-106-1-0-0">
  ...
  <dataType name="MACAddress">
    <description>All MAC addresses are ... strings of 12 hexadecimal digits ....</description>
    <string>
      <size maxLength="17"/>
      <pattern value=""/>
      <pattern value="([0-9A-Fa-f][0-9A-Fa-f]:){5}([0-9A-Fa-f][0-9A-Fa-f])"/>
    </string>
  </dataType>
  ...
</dm:document>
```

Note that the built-in primitive type (e.g. `string`) can itself contain sub-elements (referred to as facets). See Appendix I.12 for details regarding the built-in data type elements and their sub-element facets. In the example above, the string is restricted to a maximum length of 17, and conforms to a regex[10] pattern of empty <u>or</u> 6 colon-separated pairs of hex digits.

6.2.2 Define a Derived Named Data Type

A derived named data type inherits its base definition from another (inherited) named data type. This is specified using the `dataType/@base` attribute. The `base` attribute cannot reference a built-in primitive data type.

[10] See XML Schema Part 2 [12] Appendix F for details on XML Schema regular expressions.

 © The Broadband Forum. All rights reserved

To add a new data type to tr-106-1-0-0-types.xml, simply insert a `dataType` element and its related sub-elements (I.4). At a minimum, a derived data type needs to include a `name` and `base` attribute, and a `description`. The description is optional, but recommended.

The name must be unique, as it is used to identify the data type across Data Models. It must also begin with an upper-case letter in order to avoid confusion with built-in data types. The `base` attribute indicates the name of the existing named data type on which to base the new definition (i.e. the existing data type being inherited).

A derived data type is a restriction of the data type it is inheriting. Either its description will specify such restrictions, or (preferably) the data type will contain facet sub-elements that define these restrictions explicitly. See Appendix I.13 for details regarding data type facets. Common facets used within named data types include: size, range, units, pattern, and enumeration.

The following listing illustrates the definition of two named data types: IPAddress (a basic type defined with the primitive `string` type) and IPv4Address (a derived type that inherits from IPAddress). The facets within the IPv4Address data type indicate that the string has a max length of 15 (rather than 45 characters) and must conform to a specific regex[11] pattern.

```
<!DOCTYPE cwmp-datamodel-entities [
  <!ENTITY dot "\.">
  <!ENTITY octet "(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])">  <!-- 0 to 255 -->
]>

<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3" ...
             spec="urn:broadband-forum-org:tr-106-1-0-0">
  ...
  <dataType name="IPAddress">
    <description>
      IP address, i.e. IPv4 address (or IPv4 subnet mask) or IPv6 address.
    </description>
    <string>
      <size maxLength="45"/>
    </string>
  </dataType>

  <dataType name="IPv4Address" base="IPAddress">
    <description>
      IPv4 address (or subnet mask).
      Can be any IPv4 address that is permitted by the ''IPAddress'' data type.
    </description>
    <size maxLength="15"/>
    <pattern value=""/>
    <pattern value="(&octet;&dot;){3}&octet;"/>
  </dataType>
  ...
</dm:document>
```

Note – In the above example, the IPv4Address named data type is derived from an existing data type (i.e. from the IPAddress named data type). In such cases the base type restriction rules of Section A.2.3.8/TR-106 [3] must be obeyed. Base type restriction means that a valid value of a derived data type will always be a valid value for its base type.

[11] See XML Schema Part 2 [12] Appendix F for details on XML Schema regular expressions.

© The Broadband Forum. All rights reserved

6.3 Import

The `import` element (I.3) is used to import, into a local document, elements defined in other documents. By doing so, these imported elements are made visible (to be referenced) within the local document.

In other words, an element that is defined in an external document can be imported into a local document and then referenced throughout that local document (seemingly as if it were defined locally) even though the actual definition of that element does not appear directly within the local document.

Elements that can be imported:

- Bibliography (an entire bibliography, not specific items within the bibliography)
- Named data types
- Models
- Components

Each import element will specify the file and spec of the external document from which imports are being obtained (via the `import/@file` and `import/@spec` attributes, respectively)[12]. There will be one (and only one) top-level import element for each such document. Within an import element, what is actually imported from the associated document is specified by some combination of `import/dataType`, `import/model`, and `import/component` sub-elements (i.e. one for each distinct element being imported from the associated document).

Note that a document's spec value is specified in its top-level `spec` attribute. When a document is referenced via an import element, this is the corresponding value to be used in the local document's `import/@spec` attribute. However, the convention is to omit the corrigendum portion of the referenced document's spec value. For example, if the referenced document's spec value is "urn:broadband-forum-org:tr-181-2-0-1" then the value specified in the local document's `import/@spec` attribute should be "urn:broadband-forum-org:tr-181-2-0". The corrigendum is also omitted from the `import/@file` attribute value. The implication is that the external document's latest corrigendum will always be imported.

In the following example, an `import` element imports tr-069-biblio.xml (the central bibliography) into the TR-196 document. Importing a DM Instance's bibliography is a special case, where only the top-level import element need be specified.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-0" ...
             spec="urn:broadband-forum-org:tr-196-1-0-0">
  <description>...</description>

  <import file="tr-069-biblio.xml" spec="urn:broadband-forum-org:tr-069-biblio"/>
  ...
</dm:document>
```

[12] Specifying the `import/@spec` attribute is optional but recommended. It is used by the Report Tool in order to assist with validation. A mismatch between this and the external document's spec attribute is regarded as an error.

© The Broadband Forum. All rights reserved

The `import` element comes after the `description` element (if present), and before other top-level elements such as `bibliography`, `model`, and `component`.

Note – Importing a bibliography is a bit different than importing other types of items, in that a bibliography is immediately made visible within the local document via the `import` element alone. However, when importing other types of items (i.e. data types, models, components), each such item must be specified within the `import` element in order to be considered.

6.3.1 Import a Named Data Type

Note – The file tr-106-1-0-0-types.xml is the centralized document that contains named data type definitions. Generally, named data types should be defined there and be imported into other documents as needed. Defining named data types directly within other documents is not recommended.

A named data type is imported into a local document using the `import` element (I.3), which indicates the file to import from, and the `import/dataType` element, which indicates the specific data type to import. The dataType element is repeated for each named data type to be imported.

Note that once a named data type has been imported into a local document, it is visible within that document, and can be used for example in defining Data Model parameters.

The following example shows the IPAddress and MACAddress named data types being imported (from tr-106-1-0-types.xml) into the TR-104 document.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-0" ...
             spec="urn:broadband-forum-org:tr-104-1-0-0">
  <description>...</description>

  <import file="tr-069-biblio.xml" spec="urn:broadband-forum-org:tr-069-biblio"/>

  <import file="tr-106-1-0-types.xml" spec="urn:broadband-forum-org:tr-106-1-0">
    <dataType name="IPAddress"/>
    <dataType name="MACAddress"/>
  </import>
  ...
</dm:document>
```

The `import/@file` attribute indicates the name of the import file. The `import/@spec` attribute indicates the spec value of the import file. The `import/dataType/@name` attribute indicates the name of the data type to be imported from the import file.

Note – In the above example, tr-069-biblio.xml is also being imported. This is completely unrelated to importing named data types, but was included simply as an illustration of how multiple file imports appear within a document.

6.3.2 Import a Data Model

A Data Model is imported into a local document using the `import` element (I.3), which indicates the file to import from, and the `import/model` element, which indicates the specific model:version to import. The model element is repeated for each Data Model to be imported.

 © The Broadband Forum. All rights reserved

Note that once a Data Model has been imported into a local document, it is visible within that document, and can be used for example in defining the next revision of that Data Model.

In the following example the STBService:1.0 Data Model is imported into the local document, via the import/model element. It imports STBService:1.0 in order to then define the model's next revision (STBService:1.1), via the top-level model element (which needs to reference the imported model).

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3" ...
             spec="urn:broadband-forum-org:tr-135-1-1-0">
  <description>...</description>

  <import file="tr-135-1-0.xml" spec="urn:broadband-forum-org:tr-135-1-0">
    <model name="STBService:1.0"/>
  </import>
  ...
  <model name="STBService:1.1" base="STBService:1.0" isService="true">
    ...
  </model>
</dm:document>
```

The `import/@file` attribute indicates the name of the import file. The `import/@spec` attribute indicates the spec value of the import file. The `import/model/@name` attribute indicates the name and version of the model to be imported from the import file.

It is also valid to import multiple Data Models from the same file. The following example illustrates the import of both Device:1.3 and InternetGatewayDevice:1.5 from tr-157-1-0.xml.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-0" ...
             spec="urn:broadband-forum-org:tr-157-1-1-0">
  <description>...</description>

  <import file="tr-157-1-0.xml" spec="urn:broadband-forum-org:tr-157-1-0">
    ...
    <model name="Device:1.3"/>
    <model name="InternetGatewayDevice:1.5"/>
  </import>
  ...

  <model name="Device:1.4" base="Device:1.3">
    ...
  </model>

  <model name="InternetGatewayDevice:1.6" base="InternetGatewayDevice:1.5">
    ...
  </model>
</dm:document>
```

6.3.3 Import a Component

A component is imported into a local document using the `import` element (I.3), which indicates the file to import from, and the `import/component` element, which indicates the specific component to import. The component element is repeated for each component to be imported.

Note that once a component has been imported into a local document, it is visible within that document and can be used for example to reference the component's content locally.

 © The Broadband Forum. All rights reserved

The reason a component is being imported into the local document affects how the `component` element needs to be specified. There are two main reasons to import a component:

- To use it in the Data Model; i.e. insert into the local document's `model` the objects and parameters defined within the imported component.
- To update its definition (re-define it); i.e. define a <u>new</u> `component` (of the same name) based on the imported component, in order to modify its objects and parameters.

In addition, a component can also be imported into a local document simply to bring its definition forward. This is done when a previous version of the document defined the component. Doing so will include the component in the local document's namespace so a future version can import it from here without needing to know the version in which it was last modified. The convention is for a given document to import <u>all</u> components <u>defined</u> in any previous version of that document.

To Use It:

A "to use it" component import is characterized by having a `component` element with just a `name` attribute specified. In the following example three components are imported (from the TR-143 document) into the local document in this fashion. It imports these components in order to later use them within local definitions of components and/or models (not shown here; see 6.7.2).

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-1">
  <description>...</description>
  ...
  <import file="tr-143-1-0.xml" spec="urn:broadband-forum-org:tr-143-1-0">
    <component name="DownloadDiagnostics_Device2"/>
    <component name="UploadDiagnostics_Device2"/>
    <component name="UDPEchoConfig"/>
  </import>
  ...
  <model ...>
    ...
  </model>
</dm:document>
```

The `import/@file` attribute indicates the name of the import file. The `import/@spec` attribute indicates the spec value of the import file. In the "to use it" case, the `import/component/@name` attribute indicates the name of the component both as it will be referenced in the local document and as it is defined in the remote document.

When a component is imported simply using the `import/component/@name` attribute (as with the previous example), then it is referenced in the local document by the same name that it is defined by in the remote document. Such component imports cannot be updated within the local document because of a naming conflict: the name that would need to be assigned to a new version of the component is already assigned to the imported component.

 © The Broadband Forum. All rights reserved

To Re-define It:

A “to re-define it” component import is characterized by having a `component` element with both a `name` attribute and a `ref` attribute. In the following example, the UserInterface component is imported (from the TR-181i2a1 document) into the local document in this fashion. It imports this component in order to later extend it within the local document (not shown here; see 6.7.3).

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3" ...
             spec="urn:broadband-forum-org:tr-181-2-2-0">
  <description>...</description>
  ...
  <import file="tr-181-2-1.xml" spec="urn:broadband-forum-org:tr-181-2-1">
    <component name="_UserInterface" ref="UserInterface"/>
    ...
  </import>
  ...
  <model ...>
    ...
  </model>
</dm:document>
```

In the “to re-define it” case, the `import/component/@name` attribute indicates the name of the component as it is referenced in the local document, and the `import/component/@ref` attribute indicates the name of the existing component defined in the remote document. The two names must differ; the convention is to name the local (imported) component the same as the remote component but to prefix the local component’s name with an underscore.

Such component imports are free to be updated (in a new component definition) because there is no naming conflict as is found with the “to use it” case described above. How components are updated is beyond the scope of this section; see Section 6.7.3 for details.

 © The Broadband Forum. All rights reserved

6.4 Model

A Data Model contains a collection of objects and/or parameters that defines the managed objects accessible via TR-069 [1] for a CPE. There are two types: Root Data Models and Service Data Models.

The `model` element (I.8) is used to define such Data Models. Further, it is used to define both new Data Models and to define revisions to existing Data Models.

Regardless of the type of Data Model being defined, the model element will always specify the name and version of the Data Model. This is done using the `model/@name` attribute, where the name and version are represented as a single value in the form Name:Major.Minor (e.g. Device:2.0). The versioning rules outlined in Section 2.2/TR-106 [3] apply.

Note – The convention is that a document either defines components or a model (or else is a support file). A model file could define local components (whose names should begin with underscore) if it is convenient to do so.

6.4.1 Define a New Data Model

A new Data Model is the initial definition of the model; it is not based on an existing model definition. As such, it cannot have the same name and major version number as an existing Data Model.

Usually a new Data Model is given a name that is different from any of the existing Data Models, and is assigned major version 1 and minor version 0 (e.g. Something:1.0 is an entirely new Data Model). However, a new Data Model that is similar (but incompatible) with an existing Data Model can be given the same name but a different major version number (e.g. Device:2.0 is a new Data Model that is incompatible with the existing Device:1.0 Data Model). In either case, the minor version number will be 0.

Generally a new Data Model is defined within a new XML file (DM Instance document).

The following example illustrates the definition of a fictitious Something:1.0 Service Data Model (indicated by the `model/@name` attribute). Note that its major version is 1 and its minor version is 0. Also note that the `model/@isService` attribute set to *true* stipulates that it is a Service Data Model.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3" ...
             spec="urn:broadband-forum-org:tr-999-1-0-0">
  <description>...</description>

  <import file="tr-069-biblio.xml" spec="urn:broadband-forum-org:tr-069-biblio"/>
  ...
  <model name="Something:1.0" isService="true">
    ...
  </model>
</dm:document>
```

 © The Broadband Forum. All rights reserved

The following example illustrates the definition of the Device:2.0 Root Data Model (indicated by the `model/@name` attribute). Note that its major version is incremented to 2 as compared to the Device:1 Data Model, and its minor version is 0. Also note that `model/@isService` attribute has been omitted, since it defaults to false when a model is first defined.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-0">
  <description>...</description>

  <import file="tr-069-biblio.xml" spec="urn:broadband-forum-org:tr-069-biblio"/>
  ...
  <model name="Device:2.0">
    ...
  </model>
</dm:document>
```

Regardless of whether defining a new (initial) Root Data Model or Service Data Model, the `model/@name` attribute specifies a name and major version that is unique across all Data Models and its minor version should be 0.

Note – A DM Instance's `document/@spec` attribute value indicates the TR issue in which the model was published. In the listing above, Device:2.0 was initially published in TR-181 Issue 2, hence its `spec` value ends with the text *tr-181-2-0-0*.

Note – In the above examples, the central bibliography tr-069-biblio.xml is being imported. This is not absolutely necessary here, but it is recommended that all new (initial) Data Models import this bibliography file.

6.4.2 Extend an Existing Data Model

In order to extend an existing (published) Data Model, a new revision of the Data Model has to be defined that is based on the previous version.

Such revisions are generally made within a new XML file (document) that imports the previous model version. This means that the file containing the new revision will only specify the changes it is making to the Data Model and will not repeat what has already been defined in previous versions.

Each new revision has the same name and major version number as its predecessor, but its minor version number will be incremented by one.

The following example illustrates the definition of the Device:2.1 model (a revision to the Device:2.0 model). Note that the existing Device:2.0 model is first imported from a remote file in order to make it visible within the local document. Therefore the `model` element is used in two ways: to import the predecessor model, and to define the new revision.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3" ...
             spec="urn:broadband-forum-org:tr-181-2-1-0">
  <description>...</description>

  <import file="tr-181-2-0.xml" spec="urn:broadband-forum-org:tr-181-2-0">
    ...
    <model name="Device:2.0"/>
```

 © The Broadband Forum. All rights reserved

```
  </import>
  ...
  <model name="Device:2.1" base="Device:2.0">
    ...
  </model>
</dm:document>
```

In defining the new model revision, the `model/@name` attribute indicates the revision's name and version (Device:2.1), and the `model/@base` attribute indicates the predecessor's name and version (Device:2.0).

As a Data Model is revised over time, a chain of model revisions will develop where each subsequent revision is based on (and imports from) its immediate predecessor. This linkage between revisions can be illustrated with the following example:

- Device:2.0 is the initial definition of the Device:2 Data Model.
- Device:2.1 is the first revision of the Device:2 Data Model, and is based on Device:2.0.
- Device:2.2 is the second revision of the Devcie:2 Data Model, and is based on Device:2.1.
- And so on.

Note – Generally each revision to a Data Model is published within the scope of its own TR amendment. For example, Device:2.0 was initially released in TR-181 Issue 2, Device:2.1 was released in TR-181 Issue 2 Amendment 1, and Device:2.2 was released in TR-181 Issue 2 Amendment 2.

Note – For model-related documents, the `document/@spec` attribute value indicates the TR amendment in which the model was published. In the listing above, Device:2.1 was published in TR-181 Issue 2 Amendment 1, hence its `spec` value ends with the text *tr-181-2-1-0*.

6.4.3 Fixing Errata in an Existing Data Model

When errata are found within an existing (published) Data Model, such as inaccuracies with object and parameter definitions and descriptions, these mistakes can be fixed without revving the model's minor version.

This is done by republishing the document containing the flawed model definition (in its entirety) within a TR corrigendum. The new edition is a corrected copy of the flawed model, with the version number unchanged. This serves as a drop-in replacement for the defective model previously published.

The replacement model can include corrections and possibly non-functional editorial cleanup[13]. Such model replacements are generally made within a new XML file (document).

The following example illustrates the definition of the Device:2.0 replacement model. It is a corrected copy of the original Device:2.0 model. Note the `document/@spec` attribute value; its *tr-181-2-0-1* suffix indicates that this document was released with TR-181 Issue 2 Corrigendum 1.

[13] An alternative would be to defer fixing the errata until the next amendment of the associated TR, so that corrections would instead be folded in with the next planned revision to the Data Model (see 6.4.2).

 © The Broadband Forum. All rights reserved

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-1">
  <description>...</description>
  ...
  <import ...>
  ...
  <component ...>
  ...
  <model name="Device:2.0">
    ...
  </model>
</dm:document>
```

The original Device:2.0 model was defined in tr-181-2-0-0.xml (as part of TR-181 Issue 2), and its `document/@spec` attribute value ends with *tr-181-2-0-0*. The replacement Device:2.0 model was defined in tr-181-2-0-1.xml (as part of TR-181 Issue 2 Corrigendum 1), and its `document/@spec` attribute value ends with *tr-181-2-0-1*. The file name and spec value indicate that a corrigendum has been released.

The above example is highly abbreviated, but a few things should be observed:

- All of the definitions from the original document are included within the replacement document (but may vary depending on the corrections made).
- The `document/@spec` attribute value differs from the original document. For example, it indicates that the new document is tr-181-2-0-1; i.e. the portion of the number indicating corrigendum has been incremented.
- The `model/@name` attribute value is unchanged; i.e. the minor version number has not been incremented. Nor does the replacement model depend on the version of the model that it replaces. Note that the `model/@base` attribute value (not shown; only present when extending a model) will also be unchanged.

The simple rule is that the only changes will be: update the version history comment, increment the document's top-level `spec` and `file` attributes with the new corrigendum number, and make the actual fixes within the `document`.

Note – The Report Tool is corrigendum aware; its default behavior is to look for (favor) the latest corrigendum of a model. For example, TR-140 defines the StorageService:1.0 model along with two subsequent corrigenda that correct some errata (in tr-140-1-0-0.xml, tr-140-1-0-1.xml, and tr-140-1-0-2.xml). So, for a particular model version (e.g. StorageService:1.0), only the variant defined in its latest corrigendum will be considered. This should be the behavior for anything (i.e. tool, process, etc.) that ever accesses a DM Instance.

© The Broadband Forum. All rights reserved

6.5 Object (definition)

An object, and its contained parameters, defines a managed object accessible via TR-069 [1]. It is defined within a document using the `object` element (I.9). This should not be confused with the object element that appears within profiles, which is just an object reference (a reference to an object defined elsewhere) rather than an object definition.

An object can be defined within a model or a component. However, this has little bearing in how the guts of an object are defined.

Objects can be logically organized into an object hierarchy (similar to a directory tree but where each node in the hierarchy is an object). This is done in order to imply a relationship (or grouping) between objects. The hierarchy rules outlined in Section 2.1/TR-106 [3] apply.

An object's location within a hierarchy is implied by its name, which is specified by its `object/@name` attribute. Objects are named using a hierarchical form similar to a directory path. The name of a particular object 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. For example, "Device.DSL.Diagnostics." is the full path name of the Diagnostics object which sits within the DSL object which in turn sits within the Device object (the latter being the root of this particular hierarchy). An object is uniquely identified within an object hierarchy by such a path name[14]. The general name notation outlined in Section 3.1/TR-106 [3] applies. Also note that the best practice is for the name of each node in the object hierarchy to start with an uppercase letter.

Objects can be single-instance or multi-instance; the latter is often referred to as a table since each object instance can be thought of as a row (or entry) within a table. Note that all object names end with a "." (dot), whether single or multi-instance.

Note – Much of this section, examples and text, is slanted toward the initial definition of an object. Updating an existing object definition is very similar but comes with a few differences, which are discussed in Section 6.5.3.

Note – The examples throughout this section are of objects defined within models. The mechanism is almost identical in defining objects within components. The main difference with component objects is with how their full path name is resolved (see Section 6.7).

6.5.1 Defining a Single-Instance Object

A single-instance object is defined within a model or a component using the `object` element (I.9).

---

[14] Note that an object definition within a root model has its `object/@name` attribute set to a full path name, while an object definition within a service model (or component) has its `object/@name` attribute set to a partial path name. This is because the trunk of a service model (or component) is not the root of a Data Model, and so these objects must be placed somewhere within a <u>Root</u> Data Model's hierarchy before they can be used. This is done for component objects at definition time, and is done for service objects by a CPE at run-time.

 © The Broadband Forum. All rights reserved

An initial object definition includes a name, plus an indication of whether the object is read-only or read-write (from the ACS point of view), and the number of instances of that object that can exist within a CPE. This corresponds to the following required object attributes: `name`, `access`, `minEntries`, `maxEntries`.

For a single-instance object, the following restrictions apply:

- Its name ends with a "." (dot). For example, "Device.ManagementServer.".
- It has read-only access from the ACS point of view. This is indicated by the `object/@access` attribute, which is always set to "readOnly".
- There is never more than one instance of the object within a CPE. This is indicated by the `object/@minEntries` and `object/@maxEntries` attributes. The max entries attribute will always be 1. The min entries attribute will usually be 1, but can be 0 under special circumstances (see the "mutually exclusive" discussion below).

The following example illustrates the definition of the single-instance "Device." object. Note that its name ends with a dot, its access is read-only, and its max entries is 1; these are the hallmarks of a single-instance object.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-0">
  ...
  <model name="Device:2.0">
    ...
    <object name="Device." access="readOnly" minEntries="1" maxEntries="1">
      <description>The top-level object for a Device.</description>
      ...
    </object>
    ...
  </model>
</dm:document>
```

Each object should have a description. This is defined by the object/description element. In the above example, this is simply the text "*The top-level object for a Device*". However, an object description can describe in detail how the object is used, and can document additional normative requirements.

The following example illustrates the use of "*mutually exclusive*" objects (i.e. where `object/@minEntries` is set to "0"). This is when two or more such objects occur within a table definition (a multi-instance object), and where it only makes sense for one of these objects to be present in a given table entry (object instance). Also referred to as "1 of n" objects.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-0" ...
             spec="urn:broadband-forum-org:tr-069-1-0-0">
  ...
  <model name="InternetGatewayDevice:1.0">
    ...
    <object name="InternetGatewayDevice.WANDevice.{i}.WANDSLInterfaceConfig."
            access="readOnly" minEntries="0" maxEntries="1">
      ...
    </object>

    <object name="InternetGatewayDevice.WANDevice.{i}.WANEthernetInterfaceConfig."
            access="readOnly" minEntries="0" maxEntries="1">
      ...
    </object>
```

 © The Broadband Forum. All rights reserved

```
    ...
  </model>
</dm:document>
```

In the above example, we see a DSL and an Ethernet object within the WANDevice.{i} table. By defining these objects as `minEntries=0`, it allows the CPE to have only one of them present in any given WANDevice.{i} table entry. This makes sense, as we would not expect one WAN interface to support both DSL and Ethernet at the same time.

Note – An object will also contain parameter definition elements. This is not shown in the above examples. See Section 6.6 for further details.

6.5.2 Defining a Multi-Instance Object (table)

A multi-instance object (table) is defined within a model or a component using the `object` element (I.9).

An initial object definition includes a name, plus an indication of whether the object is read-only or read-write (from the ACS point of view), and the number of instances of that object that can exist within a CPE. This corresponds to the following required object attributes: `name`, `access`, `minEntries`, `maxEntries`.

For a multi-instance object, the following restrictions apply:

- Its `object/@name` attribute ends with the placeholder node name "{i}" followed by a "." (dot). For example, "Device.InterfaceStack.{i}.". In an instantiated Data Model (on an individual CPE), each placeholder is replaced by an instance identifier (i.e. an instance number or instance alias, which uniquely identifies an instance within a table).
- Its `object/@maxEntries` attribute is an integer > 1 or is the keyword "unbounded". Note that its `object/@minEntries` attribute will be an integer ≥ 0 but cannot be greater than max entries. For example, a table with min entries of 2 and max entries of 10 is valid, but min entries of 10 and max entries of 2 would be invalid.

A table can have read-only or read-write access, indicated by its `object/@access` attribute. When `object/@access` is "readOnly" the ACS can retrieve object instances from the CPE, but only the CPE can add and delete object instances. When `object/@access` is "readWrite" it is possible (makes sense) for the ACS to also be able to add and/or delete object instances.

Note – Even when `object/@access` is "readWrite", a given CPE implementation might not allow the ACS to add and delete object instances, or it might allow only add or only delete, or only delete of some instances.

For example, it does not make sense for the ACS to create an Ethernet.Interface.{i} instance or a WiFi.Radio.{i} instance (so they have `object/@access=readOnly`), but it does make sense for the ACS to create a NAT.PortMapping.{i} instance even though a given CPE implementation might not support that (so it has `object/@access=readWrite`).

 © The Broadband Forum. All rights reserved

The following listing illustrates some common forms of table definitions. Each has a name ending in "{i}." and has max entries > 1; these are the hallmarks of a multi-instance object. Note that these tables would actually have additional attributes and elements (i.e. number of entries and unique keys), but these have been omitted here; to be discussed in the next sections.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-0" ...
             spec="urn:broadband-forum-org:tr-069-1-0-0">
  ...
  <model name="InternetGatewayDevice:1.0">
    ...
    <object name="InternetGatewayDevice.DeviceInfo.VendorConfigFile.{i}."
            access="readOnly" minEntries="0" maxEntries="unbounded" ...>
      ...
    </object>

    <object name="InternetGatewayDevice.Layer3Forwarding.Forwarding.{i}."
            access="readWrite" minEntries="0" maxEntries="unbounded" ...>
      ...
    </object>

    <object name="InternetGatewayDevice.LANDevice.{i}.WLANConfiguration.{i}.WEPKey.{i}."
            access="readOnly" minEntries="4" maxEntries="4">
      ...
    </object
    ...
  </model>
</dm:document>
```

In the above example, we see the following tables defined:

- VendorConfigFile.{i} – This is a read-only, variable-sized table that can have 0 or more entries (min entries is 0, max entries is unbounded). To define a variable-sized table that allows 1 or more entries, for example, simply set `object/@minEntries` to 1. See Section 6.5.2.1 for additional information regarding such tables.
- Forwarding.{i} – This is a writable, variable-sized table that can have 0 or more entries. Its configuration is similar to the previous example, except its read-write access means that an ACS can (possibly) add and delete instances. See Section 6.5.2.2 for additional information regarding such tables.
- WEPKey.{i} – This is a fixed-sized table (min entries equals max entries). Such tables always have read-only access. See Section 6.5.2.3 for additional information regarding fixed-sized tables.

6.5.2.1 Variable-Sized Read-Only Table

A variable-sized table is simply a multi-instance object (see 6.5.2) whose max entries is greater than its min entries (i.e. attribute `object/@maxEntries > object/@minEntries`). When max entries is "unbounded" it is always regarded as being greater than min entries.

In an instantiated Data Model, the current number of instances present for each table is stored in a parameter defined in the table's parent object (i.e. one level up in the object hierarchy). The table's `object/@numEntriesParameter` attribute references this parameter. The convention is to name such parameters using the name of the table followed by the text "NumberOfEntries".

 © The Broadband Forum. All rights reserved

The following example defines the VendorConfigFile.{i} table. Its `object/@numEntries-Parameter` attribute references the table's associated "number of entries" parameter. This parameter is defined in the table's parent object (in the DeviceInfo object), and is named VendorConfigFileNumberOfEntries according to the naming rule outlined above.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-0" ...
             spec="urn:broadband-forum-org:tr-069-1-0-0">
  ...
  <model name="InternetGatewayDevice:1.0">
    ...
    <object name="InternetGatewayDevice.DeviceInfo."
            access="readOnly" minEntries="1" maxEntries="1">

      <parameter name="VendorConfigFileNumberOfEntries" access="readOnly">
        <description>{{numentries}}</description>
        <syntax>
          <unsignedInt/>
        </syntax>
      </parameter>
      ...
    </object>


    <object name="InternetGatewayDevice.DeviceInfo.VendorConfigFile.{i}."
            access="readOnly" minEntries="0" maxEntries="unbounded"
            numEntriesParameter="VendorConfigFileNumberOfEntries">
      ...
    </object>
    ...
  </model>
</dm:document>
```

Note that the `object/@numEntriesParameter` attribute value is simply the name of the referenced parameter; this value does not include the parameter's path.

Things to note about the "number of entries parameter":

- There is always one such parameter for each variable-sized table; it is always present in the table's parent object.
- It should always be named according to the rule outlined above (i.e. name of table + NumberOfEntries)
- It always has read-only access
- It always has a description of at least "`{{numentries}}`". This is a Template that the Report Tool will expand into the appropriate text (see I.2.3). Note that additional descriptive text may be included after the Template as needed.
- Its data type is always an `unsignedInt`.

Note – See Section 6.5.2.2 about tables with an "enable" parameter. While it is possible for a read-only table to have an "enable" parameter (indicating whether an entry is enabled/disabled), read-only tables do not need to specify the `object/@enableParameter` attribute.

6.5.2.2 Variable-Sized Writable Table (ACS Managed)

Defining a variable-sized writable table includes all of the same steps outlined in the previous section, in defining a variable-sized read-only table. However, there are a few additional requirements when the table's access is read-write.

 © The Broadband Forum. All rights reserved

The ACS can manage the table itself by adding and deleting object instances. A common use case is to have a new table entry be disabled until it is properly configured or until it is needed at a later time. This is accomplished by having a writable parameter within the table that indicates whether a table entry is enabled or disabled; it is often named Enable, but need not be. The table's `object/@enableParameter` attribute can reference this parameter in order to declare (and enforce) its presence.

The following example[15] defines the Forwarding.{i} table. Its `object/@numEntries-Parameter` attribute references the table's associated "number of entries" parameter. This parameter is defined in the table's parent object (in the Layer3Forwarding object). The `object/@enableParameter` attribute references the table's Enable parameter.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-0" ...
             spec="urn:broadband-forum-org:tr-069-1-0-0">
  ...
  <model name="InternetGatewayDevice:1.0">
    ...
    <object name=" InternetGatewayDevice.Layer3Forwarding."
            access="readOnly" minEntries="1" maxEntries="1">

      <parameter name="ForwardNumberOfEntries" access="readOnly">
        <description>{{numentries}}</description>
        <syntax>
          <unsignedInt/>
        </syntax>
      </parameter>
      ...
    </object>

    <object name="InternetGatewayDevice.Layer3Forwarding.Forwarding.{i}."
            access="readWrite" minEntries="0" maxEntries="unbounded"
            numEntriesParameter="ForwardNumberOfEntries" enableParameter="Enable">
      ...
      <parameter name="Enable" access="readWrite">
        <description>...</description>
        <syntax>
          <boolean/>
        </syntax>
      </parameter>
      ...
    </object>
    ...
  </model>
</dm:document>
```

Note that the table's `object/@enableParameter` attribute value is simply the name of the referenced parameter within the table; this value does not include the table's object path. The "enable" parameter definition is always contained within the table itself, and is a boolean type parameter with read-write access.

Note – According to Appendix A.2.8.1/TR-106 [3], the `enableParameter` attribute needs to be specified on writable tables (access readWrite) that have a functional unique key. Note that such tables

[15] Note that the "number of entries" parameter in this example is named ForwardNumberOfEntries (rather than the expected Forward<u>ing</u>NumberOfEntires). This is in violation of the naming convention for such parameters, but it cannot be corrected now due to backwards compatibility constraints on published Data Models.

© The Broadband Forum. All rights reserved

require enabled entries to be unique (while disabled entries need not be unique). By requiring an enable parameter here, this ensures that new entries can be properly (uniquely) configured prior to being enabled. See Section 6.5.2.4 for information regarding unique keys within tables.

6.5.2.3 Fixed-Sized Table

A fixed-sized table is simply a multi-instance object (see 6.5.2) whose min entries equals its max entries (i.e. attribute `object/@minEntries = object/@maxEntries`). As such, it always has read-only access. Since the table's number of entries is fixed, there is no need to specify the `object/@numEntriesParameter` attribute (nor does the table have an associated "number of entries" parameter defined).

The following example defines the WEPKey.{i} table. It will have exactly 4 object instances (entries) based on its min and max entries attributes.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-0" ...
             spec="urn:broadband-forum-org:tr-069-1-0-0">
  ...
  <model name="InternetGatewayDevice:1.0">
    ...
    <object name="InternetGatewayDevice.LANDevice.{i}.WLANConfiguration.{i}.WEPKey.{i}."
            access="readOnly" minEntries="4" maxEntries="4">
      ...
    </object
    ...
  </model>
</dm:document>
```

Note – See Section 6.5.2.2 about tables with an "enable" parameter. While it is possible for a read-only table to have an "enable" parameter (indicating whether an entry is enabled/disabled), read-only tables do not need to specify the `object/@enableParameter` attribute.

6.5.2.4 Unique Key for a Table

A unique key for a table is defined using the `uniqueKey` element (I.9.1). Such a key allows table entries (object instances) to be uniquely identified. A table can have zero or more such keys (though, a table with no keys defined is rare).

A unique key element (`object/uniqueKey`) references the parameters within the table that together constitute a unique key. Each parameter is referenced using an `object/-uniqueKey/parameter` element; this is simply a reference to a parameter that is defined within the same table (`object/parameter`).

The unique key elements appear after the `description` element and before the `parameter` definition elements.

In the following example we see a unique key defined within the ManageableDevice.{i} table. This key references three parameters that are defined within the table (ManufacturerOUI, SerialNumber, ProductClass). The `object/uniqueKey/parameter/@ref` attribute value is the name of a parameter being referenced from the key.

 © The Broadband Forum. All rights reserved

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-1">
  ...
  <model name="Device:2.0">
    ...
    <object name="Device.ManagementServer.ManageableDevice.{i}."
            access="readOnly" minEntries="0" maxEntries="unbounded" ...>
      <description>...</description>
      ...
      <uniqueKey>
        <parameter ref="ManufacturerOUI"/>
        <parameter ref="SerialNumber"/>
        <parameter ref="ProductClass"/>
      </uniqueKey>
      ...
      <parameter name="ManufacturerOUI" ...>
      <parameter name="SerialNumber" ...>
      <parameter name="ProductClass" ...>

      ...
    </object>
    ...
  </model>
</dm:document>
```

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.

The `object/uniqueKey/@functional` attribute specifies whether the key is functional (true) or non-functional (false). If this attribute is omitted then the key is functional by default.

In the previous example, the unique key is defined as a functional key (by default).

The following example builds on the previous example. Here we see a second key defined within the ManageableDevice.{i} table. It is a non-functional key and consists of only one parameter (Alias). Note that the Alias parameter does not relate to the purpose of the table (as is expected for a non-functional key). Both keys can be used to uniquely identify table entries.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-1">
  ...
  <model name="Device:2.0">
    ...
    <object name="Device.ManagementServer.ManageableDevice.{i}."
            access="readOnly" minEntries="0" maxEntries="unbounded" ...>
      <description>...</description>

      <uniqueKey functional="false">
        <parameter ref="Alias"/>
      </uniqueKey>

      <uniqueKey>
        <parameter ref="ManufacturerOUI"/>
        <parameter ref="SerialNumber"/>
        <parameter ref="ProductClass"/>
      </uniqueKey>
```

 © The Broadband Forum. All rights reserved

```
      <parameter name="Alias" ...>
      <parameter name="ManufacturerOUI" ...>
      <parameter name="SerialNumber" ...>
      <parameter name="ProductClass" ...>

      ...
    </object>
    ...
  </model>
</dm:document>
```

If a table's key(s) are updated in a subsequent revision to the Data Model, the key(s) need to be completely re-specified. Otherwise, it would be difficult to differentiate whether the update means to change an existing key, remove an existing key, or add an additional key.

Note – According to Appendix A.2.8.1/TR-106 [3], a writable table that has a functional unique key must also have the `enableParameter` attribute specified within the object element. Note that such tables require enabled entries to be unique (while disabled entries need not be unique). By requiring an enable parameter here, this ensures that new entries can be properly (uniquely) configured prior to being enabled.

Note – According to Appendix A.2.8.1/TR-106 [3], non-functional keys are always required to be unique, regardless of whether the table has an `enableParameter`, or is enabled or disabled. For a functional key, and if the table has an `enableParameter`, the uniqueness requirement applies only to enabled table entries.

6.5.3 Updating an Existing Object Definition

Sometimes an existing object needs to be updated. This is done using an `object` element (I.9) within a new revision of the document that defined the object in question. This implies that the object's other parent elements (e.g. `model` or `component`) will also be updated.

Note – The syntax for modifying an object is the same as for initially creating it, but there are rules. See Section A.2.10.2/TR-106 [3] for details.

An update to an object is an object definition that is based on the existing object. This is characterized by the use of the `object/@base` attribute, which indicates the path name of the existing object being updated. Note that the `object/@name` attribute, which is used in the initial definition of an object, is not used when updating an object.

The updated definition will contain changes and additions to the existing object. It should not re-define those portions of the existing object that have not changed. Generally, this means that optional object attributes that are not changing are not re-specified (e.g. `status`), and that elements within the object that are not changing are also not re-specified (e.g. `description`, `parameter`). See Appendix I.9 for a full list of attributes and elements and whether they are optional or required.

Note – Required object attributes are always specified (i.e. `access`, `minEntries`, `maxEntries`), regardless of whether or not they are being changed, because if they were not then schema validation would fail. Optional object attributes, while generally not specified unless they are changing, may be re-specified if the author deems it appropriate.

 © The Broadband Forum. All rights reserved

Common reasons to update an object include:

- Change an attribute value; e.g. set status to deprecated or deleted
- Update its description
- Add or update one of its parameters
- Add a unique key to a table

The following example illustrates an update of the IP Interface object (the first listing is the initial object definition in tr-181-2-0-1 and the second listing is its update in the much later tr-181-2-2-0). The object was initially defined with a description, a unique key, and several parameters (including Enable and Status). The object update changes the description, changes the existing Enable parameter, and adds new parameters (including IPv4Enable). As expected, the update is within a new document revision and the object's parent elements are also updated (in this case, its model).

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-1">
  ...
  <model name="Device:2.0">
    ...
    <object name="Device.IP.Interface.{i}." access="readWrite"
            numEntriesParameter="InterfaceNumberOfEntries" enableParameter="Enable"
            minEntries="0" maxEntries="unbounded">

      <description>IP interface table (a stackable interface object...</description>

      <uniqueKey ...>
        ...
      </uniqueKey>
      ...

      <parameter name="Enable" ...>
        ...
      </parameter>

      <parameter name="Status" ...>
        ...
      </parameter>

      ...
    </object>
    ...
  </model>
</dm:document>
```

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3" ...
             spec="urn:broadband-forum-org:tr-181-2-2-0">
  ...
  <model name="Device:2.2" base="Device:2.1">
    ...
    <object base="Device.IP.Interface.{i}." access="readWrite"
            minEntries="0" maxEntries="unbounded">

      <description action="append">Each IP interface can be attached to the...</description>

      <parameter base="Enable" ...>
        ...
      </parameter>

      <parameter name="IPv4Enable" ...>
        ...
      </parameter>
```

 © The Broadband Forum. All rights reserved

```
      ...
    </object>
    ...
  </model>
</dm:document>
```

The above example is highly abbreviated, but a few things should be observed within the updated document in the second listing:

- The `object/@base` attribute is used to reference the existing object definition.
- All of the object's required attributes are present (though none were changed).
- None of the object's optional attributes are present, since they have not changed. Only changed attributes need be specified.
- The object description is present because it is being updated. Had it not been updated, then it would have been omitted.
- The object's unique key is omitted since it has not changed. Had any part of the unique key(s) been changed, it would be re-specified in its entirety.
- The existing parameters that are not being updated have been omitted (e.g. Status). The existing parameters that are being updated are present (e.g. Enable).
- New parameters are defined within the updated object (e.g. IPv4Enable).

Note – In the above example, it is not shown in the second listing, but the previous Device:2.1 model must be imported before it can be used to define the updated Device:2.2 model. See 6.4.2 for details on updating an existing model.

When an existing object needs to be deprecated, obsoleted, or deleted, this is done by updating its `object/@status` attribute. The object's description must also be updated to include an explanation of why the status was changed. See Section 6.10 for details. Note that such a status update will also apply to the object's contained items (e.g. child objects and parameters), but only if this will "promote" a given item's status to a "higher" value[16].

[16] When an item's status is modified it must be a "promotion" 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.

 © The Broadband Forum. All rights reserved

6.6 Parameter (definition)

A parameter represents part of a CPE's configuration or status, accessible via TR-069 [1]. It is defined within a DM Instance document using the parameter element (I.10). This should not be confused with the parameter element that appears within profiles, which is just a parameter reference (a reference to a parameter defined elsewhere) rather than a parameter definition.

A parameter can be defined within a model[17], a component, or an object. However, this has little bearing in how the guts of a parameter are defined.

Parameters are most often defined within the scope of an object. When they are defined outside an object, they are referred to as top-level parameters and appear within a model or component before object definitions.

A parameter is uniquely identified by its full path name. This is the concatenation of the full path name of its parent object (if any)[18], plus the name of the parameter itself (which is specified by its `parameter/@name` attribute). For example, "Device.ManagementServer.EnableCWMP" is the full path name of the EnableCWMP parameter which sits within the ManagementServer object. The general name notation outlined in Section 3.1/TR-106 [3] applies.

Each parameter is defined in terms of its data type; e.g. boolean, string, etc. This is specified within the `parameter/syntax` element.

Note – Much of this section, examples and text, is slanted toward the initial definition of a parameter. Updating an existing parameter definition is very similar but comes with a few differences, which are discussed in Section 6.6.7.

Note – Parameter values are not present within DM Instance documents; these documents contain syntax definitions not run-time configurations. A parameter definition stipulates the valid syntax of a parameter's value (i.e. the type and range of data permitted) rather than the actual run-time value itself.

Note – The examples throughout this section are of parameters defined within models. The mechanism is almost identical in defining parameters within components. The main difference with component parameters is with how their full path name is resolved (see Section 6.7).

6.6.1 Defining a Parameter (The Basics)

A parameter is defined within a model, component, or object using the `parameter` element (I.10).

An initial parameter definition includes a name, plus an indication of whether the parameter is read-only or read-write (from the ACS point of view), and the syntax for valid parameter values. This corresponds to the following <u>required</u> attributes and elements: `name`, `access`, `syntax`.

[17] The only occurrence of this is a NumberOfEntries parameter for a Service Data Model.
[18] Object path names are discussed in Section 6.5.

 © The Broadband Forum. All rights reserved

The following restrictions also apply:

- Its name does not include a path (the `parameter/@name` attribute value does not permit dot notation). Also note that a parameter name should start with an uppercase letter.
- Its access can be read-only or writable (the `parameter/@access` attribute value is one of: readOnly or readWrite).
- Its syntax can be defined either using one of the built-in primitive data types or using a named data type. These two options are discussed in Sections 6.6.1.1 and 6.6.1.2 below.

The following example illustrates the definition of the EnableCWMP parameter. It is a writable parameter that is defined within the scope of the ManagementServer object. Its full path name is Device.ManagementServer.EnableCWMP. Note that this is an incomplete example since the syntax details have been omitted (to be discussed in the next sections).

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-0">
  ...
  <model name="Device:2.0">
    ...
    <object name="Device.ManagementServer." ...>
      ...
      <parameter name="EnableCWMP" access="readWrite">
        <description>Enables and disables the CPE's support for CWMP...</description>
        <syntax>
          ...
        </syntax>
      </parameter>
      ...
    </object>
    ...
  </model>
</dm:document>
```

Each parameter should have a description. This is defined by the `parameter/description` element. In the above example, the description begins with the text "*Enables or disables the CPE's support for CWMP*". A parameter description should describe in detail how the parameter is used and document additional normative requirements.

6.6.1.1 Syntax Using a Built-In Primitive Data Type

A parameter value's valid syntax is defined using the parameter `syntax` element (I.10.1). This element can specify a parameter's data type and range of permitted values.

Note – There are two ways to define a parameter's data type: either via a built-in primitive data type, or via a reference to a named data type (6.6.1.2). They are mutually exclusive. This section discusses the use of primitive data types.

There is a set of built-in primitive data type elements that can be used in defining a parameter's syntax. These are: `base64`, `boolean`, `dateTime`, `hexBinary`, `int`, `long`, `string`, `unsignedInt`, `unsignedLong` (see Section I.12 for details on each). One of these elements will appear within the parameter's syntax element (e.g. `parameter/syntax/boolean`); a parameter's syntax element can only contain one such primitive data type element.

© The Broadband Forum. All rights reserved

The following example illustrates the definition of a parameter with a boolean data type. This is the same EnableCWMP parameter that was defined earlier, except now the `parameter/-syntax/boolean` element is shown. Note that only one primitive data type element appears within the syntax element.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-0">
  ...
  <model name="Device:2.0">
    ...
    <object name="Device.ManagementServer." ...>
      ...
      <parameter name="EnableCWMP" access="readWrite">
        <description>Enables and disables the CPE's support for CWMP...</description>
        <syntax>
          <boolean/>
          ...
        </syntax>
      </parameter>
      ...
    </object>
    ...
  </model>
</dm:document>
```

Note – A data type definition within a parameter is referred to as an anonymous data type. An anonymous data type can only apply to the parameter within which it is defined. See Section A.2.3.2/TR-106 [3] for additional discussion.

In the above example, the built-in type specified is `parameter/syntax/boolean`. Defining a parameter using one of the other built-in types is as simple as replacing this element with one of the other built-in element types (e.g. `parameter/syntax/string`).

6.6.1.2 Syntax Using a Named Data Type

A parameter value's valid syntax is defined using the parameter `syntax` element (I.10.1). This element can specify a parameter's data type and range of permitted values.

Note – There are two ways to define a parameter's data type: either via a built-in primitive data type (6.6.1.1), or via a reference to a named data type. They are mutually exclusive. This section discusses the use of named data types.

A named data type should be used when there is a set of data requirements that can apply to a range of parameters. For example, an IP address or a MAC address. By defining and using a named data type, it guarantees that these data requirements will be applied consistently. See Section 6.2 for details on defining named data types.

A parameter can reference a named data type using the `parameter/syntax/dataType` element (I.10.3). The reference is made via its `ref` or `base` attribute. The named data type must already be defined elsewhere (and possibly imported into the local document) before it can be referenced from the parameter definition.

 © The Broadband Forum. All rights reserved

The `dataType/@ref` attribute will reference a named data type when the data type is used as is (without change). If the parameter's referenced data type needs to be altered (via facets), then instead the `dataType/@base` attribute is used to reference the named data type (see Section 6.6.1.3 for details on refining a data type using facets).

The following example illustrates the definition of a parameter with a named data type. The `parameter/syntax/dataType/@ref` attribute references the IPAddress data type. Note that only one data type element appears within the `syntax` element; and since the `dataType/@ref` attribute is used, facets are not permitted within the `dataType` element.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-0">
  ...
  <model name="Device:2.0">
    ...
    <object name="Device.IP.ActivePort.{i}." ...>
      ...
      <parameter name="RemoteIPAddress" access="readOnly">
        <description>...</description>
        <syntax>
          <dataType ref="IPAddress"/>
        </syntax>
      </parameter>
      ...
    </object>
    ...
  </model>
</dm:document>
```

6.6.1.3 Refining a Data Type Using Facets

A parameter's data type definition can be refined using facets. Facet elements are used within primitive and named data types to specify different aspects of the data type being defined, such as string size, numeric range, etc.

The full set of facet elements is: `size`, `range`, `units`, `pattern`, `enumeration`, `enumerationRef`, `instanceRef`, `pathRef`. See Section I.13 for details on using each of these facets. Additional guidance in defining parameters using reference-based facets is also provided in Section 6.6.6.

Note that not all facets can be used within all primitive data types. This is also the case within named data types, since they are derived from primitive data types. Section I.12 Table 40 lists which facets are valid with each primitive data type. Note that the `boolean` and `dateTime` types have no available facets.

Facet use within parameters is much the same regardless of whether the parameter has a primitive or named data type. However, when referencing a named data type, the parameter's `dataType/@base` attribute must be specified rather than the `dataType/@ref` attribute (i.e. use of the `base` attribute indicates that the referenced named data type will be altered via facets, while use of the `ref` attribute indicates that the named data type is referenced as is without facets).

© The Broadband Forum. All rights reserved

The following example illustrates some common definitions using facets (within parameters that have a built-in primitive data type). The first parameter is a string type with maximum length of 256 (uses the `size` facet). The second parameter is an unsigned integer type with range 1 to 65535, whose value will be measured in seconds (uses multiple facets: `range` and `units`).

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-0">
  ...
  <model name="Device:2.0">
    ...
    <object name="Device.ManagementServer." access="readOnly" minEntries="1" maxEntries="1">
      ...
      <parameter name="URL" access="readWrite">
        <description>...</description>
        <syntax>
          <string>
            <size maxLength="256"/>
          </string>
        </syntax>
      </parameter>

      <parameter name="CWMPRetryMinimumWaitInterval" access="readWrite">
        <description>...</description>
        <syntax>
          <unsignedInt>
            <range minInclusive="1" maxInclusive="65535"/>
            <units value="seconds"/>
          </unsignedInt>
        </syntax>
      </parameter>
      ...
    </object>
    ...
  </model>
</dm:document>
```

And the next example demonstrates the use of a facet within a parameter that has a named data type (i.e. within the `parameter/syntax/dataType` element)[19]. The example assumes that String255 is a named data type that has been defined as a string with max length 255. Use of the `dataType/@base` attribute (rather than `ref` attribute) indicates that the referenced named data type will be altered via facets. The `size` facet is then used to restrict the parameter's max string length down to 127 characters.

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

Note – In the above example, the parameter's anonymous data type is derived from an existing data type (i.e. from the String255 named data type). In such cases the base type restriction rules of Section A.2.3.8/TR-106 [3] must be obeyed. Base type restriction means that a valid value of a new data type will always be a valid value for its base type.

[19] This is a made-up example since it is a scenario not encountered in the published Data Models.

© The Broadband Forum. All rights reserved

Certain facets can appear multiple times within a parameter definition, in order to further refine the parameter's data type. Such facets are: `size`, `range`, `enumeration`, and `pattern`. Multiple size facets are used to indicate different string-length ranges. Multiple range facets are used to define disjoint integer ranges. Multiple enumeration facets are used to define a set of valid string values. Multiple pattern facets are used to specify a set of valid string-value patterns.

The following example illustrates several parameters where a particular facet appears multiple times in its definition. The WEPKey parameter has two valid string length ranges (exactly 5 bytes and exactly 13 bytes). The MSC parameter has two valid integer ranges (the second range being optionally supported by a CPE). The ManufacturerOUI parameter defines a string type that only allows values that conform to two patterns (empty string or a six-digit string of hex characters). The SupportedModes parameter is a string type that only allows values that match one of its enumerations (i.e. Send, Receive, or Both).

```
<parameter name="WEPKey" access="readWrite">
  <description>...</description>
  <syntax>
    <hexBinary>
      <size minLength="5" maxLength="5"/>
      <size minLength="13" maxLength="13"/>
    </hexBinary>
  </syntax>
</parameter>

<parameter name="MSC" access="readWrite">
  <description>...</description>
  <syntax>
    <int>
      <range minInclusive="-1" maxInclusive="15"/>
      <range minInclusive="16" maxInclusive="31" optional="true"/>
    </int>
  </syntax>
</parameter>

<parameter name="ManufacturerOUI" access="readOnly">
  <description>Unique identifier of the associated gateway. {{pattern}} ...</description>
  <syntax>
    <string>
      <pattern value=""/>
      <pattern value="[0-9A-F]{6}"/>
    </string>
  </syntax>
</parameter>

<parameter name="SupportedModes" access="readOnly">
  <description>The supported RIP protocol modes. {{enum}} ...</description>
  <syntax>
    <string>
      <enumeration value="Send"/>
      <enumeration value="Receive"/>
      <enumeration value="Both"/>
    </string>
  </syntax>
</parameter>
```

Note – Enumeration-valued parameters can include an `{{enum}}` Template in their description. This will be replaced by the Report Tool (when it generates an HTML report) with boilerplate text that describes the set of valid enumerations (see I.2.3). In the absence of an `{{enum}}` Template, the `{{noenum}}` Template can be used to suppress any default boilerplate text that the Report Tool might

 © The Broadband Forum. All rights reserved

generate. For pattern-valued parameters, the same holds true with `{{pattern}}` and `{{nopattern}}`.

6.6.1.4 Default Value

A default value is specified for a parameter using the `parameter/syntax/default` element (I.10.4). This element can indicate either a factory default or an object default.

An object default should only be used with parameters that are created when the ACS uses the AddObject RPC to create an instance. A factory default can potentially be used with any parameter, but only applies when the parameter's default value is based on some standard, e.g. an IETF RFC. Also, if there is a factory default then it also acts as an object default.

The type of default is indicated by the required attribute `parameter/syntax/-default/@type`. Possible values for this attribute are: factory or object. The default value itself is indicated by the required attribute `parameter/syntax/default/@value`. This value must be valid for the parameter's data type.

The following example illustrates a parameter with a factory default. This is the same EnableCWMP parameter as in the example from Section 6.6.1.1, except now the `parameter/syntax/default` element is specified. Note that the default value (true) is valid for the parameter's boolean data type.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-0">
  ...
  <model name="Device:2.0">
    ...
    <object name="Device.ManagementServer." access="readOnly" minEntries="1" maxEntries="1">
      ...
      <parameter name="EnableCWMP" access="readWrite">
        <description>Enables and disables the CPE's support for CWMP...</description>
        <syntax>
          <boolean/>
          <default type="factory" value="true"/>
        </syntax>
      </parameter>
      ...
    </object>
    ...
  </model>
</dm:document>
```

The following example illustrates a parameter with an object default. This is a legitimate use of an object default since the RekeyingInterval parameter comes about due to the creation of the AccessPoint.{i} object. Note that the default value (3600) is valid for the parameter's unsigned integer data type.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-0">
  ...
  <model name="Device:2.0">
    ...
    <object name="Device.WiFi.AccessPoint.{i}.Security." access="readOnly"
            minEntries="1" maxEntries="1">
```


© The Broadband Forum. All rights reserved

```
      ...
      <parameter name="RekeyingInterval" access="readWrite">
        <description>Enables and disables the CPE's support for CWMP...</description>
        <syntax>
          <unsignedInt>
          <default type="object" value="3600"/>
        </syntax>
      </parameter>
      ...
    </object>
    ...
  </model>
</dm:document>
```

Note that the default element if present must be the last element within the syntax.

6.6.1.5 Active Notify and Forced Inform

Note – See TR-069 [1] for explanation of the Active Notification and Forced Inform mechanisms.

The `parameter/@activeNotify` attribute is used to define a parameter with active notification restrictions. This is an optional attribute; if it is omitted when first defining the parameter it will default to normal (i.e. normal notification capabilities). On subsequent updates to the parameter definition, this attribute can be omitted (assuming its value does not need to change).

Possible values for the `activeNotify` attribute are:

- normal – Indicates parameter has no limitations with respect to active notifications.
- forcedEnabled – Indicates parameter will always have active notification set.
- forcedDefaultEnabled – Indicates parameter will have active notification set by default, but that it can be disabled.
- canDeny – Indicates that the CPE can deny a request by the ACS to set active notification on the parameter.

The `parameter/@forcedInform` attribute is used to define a parameter that will be included in all CWMP Inform messages (i.e. a forced inform parameter). This is an optional attribute; if it is omitted when first defining the parameter it will default to false (i.e. not forced inform). On subsequent updates to the parameter definition, this attribute can be omitted (assuming its value does not need to change).

Possible values for the `forcedInform` attribute are true (forced inform) or false (not forced inform).

The following example illustrates a parameter with active notify set to canDeny and forced inform set to true.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-0">
  ...
  <model name="Device:2.0">
    ...
    <object name="Device.ManagementServer." access="readOnly" minEntries="1" maxEntries="1">
      ...
```


© The Broadband Forum. All rights reserved

```
      <parameter name="ParameterKey" access="readOnly"
                 activeNotify="canDeny" forcedInform="true">
        ...
      </parameter>
      ...
    </object>
    ...
  </model>
</dm:document>
```

Note – The `activeNotify` and `forcedInform` attributes are often omitted when first defining a parameter, since their default values (i.e. normal and false) are correct for most new parameter definitions.

6.6.2 Number-of-Entries Parameter

Each variable-sized table (multi-instance object) requires an associated “number of entries” parameter, which is defined in the table’s parent object (i.e. one level up in the object hierarchy). This parameter will indicate the current number of instances present in the table.

The convention is to name such parameters using the name of the associated table followed by the text “NumberOfEntries”.

Each “number of entries” parameter is read-only and has an `unsignedInt` data type. Its description need only be `{{numentries}}`; this is a description Template which the Report Tool will replace with boilerplate text (when it generates an HTML report) that denotes the associated table (see I.2.3). This text will be something like “The number of entries in the <table> table.”.

The following example snippet defines the VendorConfigFileNumberOfEntries parameter. Based on its name, the implication is that its associated table is VendorConfigFile.{i}.

```
      <parameter name="VendorConfigFileNumberOfEntries" access="readOnly">
        <description>{{numentries}}</description>
        <syntax>
          <unsignedInt/>
        </syntax>
      </parameter>
```

Note – The associated table, not shown here, also requires an `object/@numEntriesParameter` attribute that explicitly references the “number of entries” parameter. See Section 6.5.2.1 for details.

6.6.3 Hidden-Valued Parameter

A hidden-valued parameter is a parameter whose value is hidden from the ACS on read back. It is defined by specifying the `parameter/syntax/@hidden` attribute as true (I.10.1). This type of parameter is desired when its value will contain sensitive information, for example, a password or WEP key.

 © The Broadband Forum. All rights reserved

A hidden-valued parameter will always read back as the null value for the parameter's base data type (see Annex 2.3.5/TR-106 [3]). In summary, each primitive data type has an associated null value. 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)

The best practice is to specify the `parameter/syntax/@hidden` attribute as true when first defining the parameter. On subsequent updates to the parameter definition, this attribute can be omitted (assuming its value does not need to change).

The following example defines a hidden-valued parameter, named Password. It is hidden because the attribute `parameter/syntax/@hidden=true`. Since its primitive data type is string, it will read back as an empty string regardless of its actual parameter value.

```
<parameter name="Password" access="readWrite">
  <description>...</description>
  <syntax hidden="true">
    <string>
      <size maxLength="256"/>
    </string>
  </syntax>
</parameter>
```

6.6.4 Command Parameter

A command parameter is a parameter that is associated with a CPE action; setting the parameter triggers the action. It is defined by specifying the `parameter/syntax/@command` attribute as true (I.10.1).

Such a parameter is not part of the device configuration and will always read back as the null value for the parameter's base data type (see Annex 2.3.5/TR-106 [3]). In summary, each primitive data type has an associated null value. 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)

The best practice is to specify the `parameter/syntax/@command` attribute as true when first defining the parameter. On subsequent updates to the parameter definition, this attribute can be omitted (assuming its value does not need to change).

The nature of a command parameter means that it will be writable and will typically have a boolean data type.

© The Broadband Forum. All rights reserved

The following example defines a command parameter, named PasswordReset. It is a command parameter because the attribute `parameter/syntax/@command=true`. It has a typical access and syntax for a command parameter.

```
<parameter name="PasswordReset" access="readWrite">
  <description>...</description>
  <syntax command="true">
    <boolean/>
  </syntax>
</parameter>
```

6.6.5 List-Valued Parameter

A list-valued parameter is specified using the `parameter/syntax/list` element (I.10.1/Table 27 and I.10.2); it indicates that the parameter value is a list of items. This is referred to as a comma-separated list in Section 3.2.3/TR-106 [3]. Such a parameter is always a string, and its data type specification applies to individual list items rather than the parameter value as a whole.

The `parameter/syntax/list` element must be the first element within the syntax, followed by the data type specification. The items' data type is either defined as a primitive data type or a named data type (see Sections 6.6.1.1 and 6.6.1.2, respectively, for details). Note that the data type elements used here are the same as those used with non-list-valued parameter definitions.

The number of items permitted in a list is specified using the `list` element's `minItems` and `maxItems` attributes. Both are optional attributes; neither, either, or both can be specified. When first defining a list-valued parameter, these attributes default to 0 and unbounded (respectively) if they are not specified and there is no implied alternative.

As noted, the parameter value as a whole is always a string. The `parameter/syntax/-list/size` facet element can be used to explicitly specify the minimum and maximum length of this string value, via its `minLength` and `maxLength` attributes (see I.13.1). Alternatively, the maximum length can instead be implied by the maximum number of items and the individual item lengths (as discussed in Section 3.2.6/TR-106 [3]). The latter is preferred when the number and length of items is known.

Strictly speaking, it is not necessary to specify a list's max items or max length, but indicating some sort of maximum sizing information is good practice. If there is no explicit or implied maximum length, the default maximum is 16 characters unless otherwise indicated in an associated parameter description.

Note – List-valued parameters can include a `{{list}}` Template in their description. This will be replaced by the Report Tool (when it generates an HTML report) with boilerplate text that describes the list's characteristics (see I.2.3). In the absence of a `{{list}}` Template, the `{{nolist}}` Template can be used to suppress any default boilerplate text that the Report Tool might generate.

<u>Example – list with max length</u>:

© The Broadband Forum. All rights reserved

The following example defines a parameter whose value is a list of language codes. The individual items are strings, as indicated by the `parameter/syntax/string` element. The number of possible list items is not specified, so the defaults are assumed (i.e. between 0 and an unbounded upper range). Since the max number of items is unbounded, the parameter value's max length cannot be inferred; hence it is defined explicitly (i.e. it has a minimum length of 0, by default, and a specified maximum of 256 characters).

```
<parameter name="AvailableLanguages" access="readOnly">
  <description>{{list}} List items represent user-interface languages...</description>
  <syntax>
    <list>
      <size maxLength="256"/>
    </list>
    <string/>
  </syntax>
</parameter>
```

Note that while the max number of items defaults to unbounded, in practice there is an upper limit based on the 256 character max length and the fact that items are comma-delimited within the parameter value.

Example – list with max items (and no max length given):

The following example defines a parameter whose value is a list of MAC addresses (as defined by the named data type MACAddress). The list's max number of items is 16. The list's max length is not specified, but it is implied given its max items and the fact that a MACAddress has a max length.

```
<parameter name="EndStationMACs" access="readOnly">
  <description>{{list}} List items represent MAC addresses of end stations...</description>
  <syntax>
    <list maxItems="16"/>
    <dataType ref="MACAddress">
  </syntax>
</parameter>
```

In the above example, the list's max length is at least 287 characters (i.e. 16 max items, and each item has a max length of 17, plus a comma-separator between each item).

Example – enumeration list:

The following example defines a parameter whose value is a list of enumeration items. As always, the `parameter/syntax/list` element appears first within the syntax followed by the data type specification for the individual items. With an enumeration list, the max number and length of items is implied (i.e. we know how many different enumerations there are and the length of each), so there is no need to explicitly define the list's max length or max number of items (it can be derived).

```
<parameter name="PossibleConnectionTypes" access="readOnly">
  <description>{{list}} List items indicate the types of connections...</description>
  <syntax>
    <list/>
```


© The Broadband Forum. All rights reserved

```
          <string>
            <enumeration value="Phone"/>
            <enumeration value="Coax"/>
          </string>
        </syntax>
      </parameter>
```

In the above example, there are two enumeration values; so the list's max items is 2, and its max length is at least 10 characters given the length of the two items and a comma separator. The min items is 0 by default. Note that if there are vendor-specific enumerations defined, then the string max length might be longer.

Example – fixed-sized list:

The following example defines a parameter with a fixed-sized list; it has exactly 8 items where each item is an unsigned integer between 0 and 7 (e.g. a string value containing a comma-separated list such as `"1,2,0,5,2,8,9,2"`).

```
      <parameter name="PriorityRegeneration" access="readWrite">
        <description>{{list}} List items represent user priority regeneration...</description>
        <syntax>
          <list minItems="8" maxItems="8"/>
          <unsignedInt>
            <range minInclusive="0" maxInclusive="7"/>
          </unsignedInt>
          ...
        </syntax>
      </parameter>
```

6.6.6 Reference Parameter

A reference parameter references another parameter or object. This is defined using a reference facet (see Annex A.2.3.7/TR-106 [3]) within the parameter's data type specification. There are three kinds of reference:

- **Path reference**: references another parameter or object via a path name (see Section 6.6.6.1). The value of a path-reference parameter represents a specific path name.
- **Instance reference**: references an object instance (table row) via an instance number (see Section 6.6.6.2). The value of an instance-reference parameter represents a specific instance number.
- **Enumeration reference**: references a list-valued parameter via a path name (see Section 6.6.6.3). The current value of the referenced parameter indicates the valid enumerations for the enumeration-reference parameter; i.e. the value of the enumeration-reference parameter is one of the referenced list items or a specified "null" value.

All such parameters need to specify the path name of the parameter or object that they reference. For instance- and enumeration-reference parameters, the specific path name referenced is indicated by the parameter definition itself; for path-reference parameters, the parameter definition might narrow the set of possible path names while its run-time parameter value actually specifies the specific path name referenced.

 © The Broadband Forum. All rights reserved

These path names can be full path names or relative path names, as discussed in Annex A.2.3.4/TR-106 [3]. 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.

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. There are three kinds of scope: normal, model, and object. The preference is to use normal, a hybrid scope which usually gives the desired behavior based on the format of the path name. Another benefit of using normal scope is that any leading hash characters ("#") in the path name will redirect it up one level to the parent of the current object (or the parent's parent for two hash characters, and so on)[20]. Path name scope is discussed in Annex A.2.3.4/TR-106 [3].

Note that an enumeration reference is fundamentally different from the other two kinds of reference. With path- and instance-reference parameters, the parameter value indicates which specific parameter or object is being referenced (and will have additional reference info within its definition). However, an enumeration-reference parameter maintains this reference info solely within its definition, while its parameter value will be one of the list items from the current value of the referenced parameter.

Note – Reference parameters can include a `{{reference}}` Template in their description. This will be replaced by the Report Tool (when it generates an HTML report) with boilerplate text that describes the parameter or object being referenced (see I.2.3). In the absence of a `{{reference}}` Template, the `{{noreference}}` Template can be used to suppress any default boilerplate text that the Report Tool might generate.

6.6.6.1 Path-Reference Parameter

A path-reference parameter references another parameter or object via its path name.

The reference is defined using a `pathRef` facet element (I.13.3) within the referencing parameter's data type specification. Since the parameter's value will be a specific path name, the use of pathRef only applies to string and its derived types.

The `pathRef` element has several attributes that aid in specifying which parameters and objects will be targeted as candidates for the reference. These are: `targetParent`, `targetParentScope`, `targetType`, and `targetDataType`. The most commonly used targeting attributes are `targetParent` and `targetType`.

The set of all parameters and objects that can be referenced is restricted using the optional `pathRef/@targetParent` attribute. This attribute can contain a list of path names indicating which objects to look under; only the immediate children of one of these specified parent objects can be referenced. If the list is empty (the default), then anything can be referenced.

[20] The path name is always relative to the current object. The # provides a way for it to reference its parent.

 © The Broadband Forum. All rights reserved

Note – A "{i}" placeholder in a targetParent path name acts as a wild card, and can therefore represent multiple parent objects. The targetParent path name(s) cannot contain explicit instance numbers[21].

The targetParent's path name scope is specified using the optional `pathRef/-@targetParentScope` attribute (with possible values: normal, model, object), however, this should be avoided when possible! The best practice is to use normal scope, the default if this attribute is omitted when first defining the parameter. Unfortunately there are some cases where you need to set the scope differently, e.g. where the path name begins with "Device.".

Note – Normal is a hybrid scope which usually gives the desired behavior (and where the scope of a path name is implied by the format of the path name itself). If a targetParent path begins with "Device." or "InternetGatewayDevice." then it is relative to the top of the naming hierarchy. If a targetParent path begins with a dot then it is relative to the Root or Service Object. Otherwise, a targetParent path is relative to the current object (and any leading hash characters in a targetParent path will redirect it up one level to the parent of the current object, or the parent's parent, and so on for each hash character). Path names and scope are discussed in more detail in Annex A.2.3.4/TR-106 [3].

The type of item (parameter or object) that can be referenced is specified using the optional `pathRef/@targetType` attribute. One of:

- **any**: either a parameter or an object can be referenced (default)
- **parameter**: only a parameter can be referenced
- **object**: any type of object can be referenced
- **single**: only a single-instance object can be referenced
- **table**: only a multi-instance object (table) can be referenced
- **row**: only a multi-instance object instance (table row) can be referenced

When the item to be referenced is a parameter, then the type of parameter to be referenced can be restricted using the optional `pathRef/@targetDataType` attribute. This is relevant only when targetType is configured for parameters (i.e. any or parameter). Possible values for targetDataType are: any (any data type), base64, boolean, dateTime, hexBinary, integer (any numeric data type), int, long, string, unsignedInt, unsignedLong, or the name of some named data type. The default is any if not specified when first defining the parameter.

The reference type, either strong or weak, is specified using the `pathRef/@refType` attribute. This is the only required attribute for the `pathRef` element. Reference type is discussed in Section A.2.3.6/TR-106 [3]. In summary, a strong reference always either references a valid parameter or object, or else is a null reference. A weak reference does not necessarily reference an existing parameter or object.

Note – A null reference indicates that a referencing 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. an empty string when using pathRef since it is string-based). See Section A.2.3.5/TR-106 [3].

---

[21] This is the exact opposite to how a path name is specified within the reference parameter's value. The actual referenced path name cannot contain "{i}" placeholders but may contain explicit instance numbers as needed. This difference is the distinction between a Data Model definition (where instance numbers are not known) and an instantiated run-time Data Model (where instance numbers are needed). See Section 3.2.4/TR-106 [3].

 © The Broadband Forum. All rights reserved

Note – If the parameter is actually a list of references, then there is no explicit null value. Instread, an empty list indicates no reference.

The following example defines a reference parameter using a `pathRef` facet. The `targetType` attribute indicates that it will reference a table row, and the `targetParent` attribute indicates two tables that this row can come from. Note that `targetParent` contains relative path names here, and since they are prefixed with a hash character, they are redirected to the Device.Bridging object one level up in the naming hierarchy (i.e. the targetParent path names resolve to Device.Bridging.Bridge. and Device.Bridging.Bridge.{i}.VLAN. respectively) . This is a strong reference (refType is strong), so the parameter's value will either be the path name of a valid object instance or an empty string.

```
<object name="Device.Bridging.Filter.{i}." ...>
  ...
  <parameter name="Bridge" access="readWrite">
    <description>{{reference}} ...</description>
    <syntax>
      <string>
        <size maxLength="256"/>
        <pathRef refType="strong"
                 targetParent="#.Bridge. #.Bridge.{i}.VLAN." targetType="row"/>
      </string>
    </syntax>
  </parameter>
  ...
</object>
```

In the above example, the string has a max length of 256 characters. This is a common practice with path-reference parameters. If no max length were specified, then the default would have been 16 characters which would likely be insufficient.

The following example defines a reference parameter named Reference. The `targetType` attribute indicates that it will reference a parameter, and the lack of a `targetParent` attribute indicates that any parameter can be referenced (i.e. no restriction for which parameter to reference, just so long as that parameter is defined). Since this is a reference to a parameter, the `targetDataType` attribute could have been employed but in this case was not desired. This is a weak reference (refType is weak), so the parameter's value need not be a path name to an existing parameter.

```
<parameter name="Reference" access="readWrite">
  <description>{{reference}} ...</description>
  <syntax>
    <string>
      <size maxLength="256"/>
      <pathRef refType="weak" targetType="parameter"/>
    </string>
  </syntax>
</parameter>
```

The following example defines a parameter that combines a path-reference with a list, so its value will be a list of path references. The `targetType` attribute indicates that each list item will reference a table row, and the `targetParent` attribute indicates which table these rows will come from. Note that targetParent contains a relative path name here, and since it is prefixed with a dot, it is

© The Broadband Forum. All rights reserved

relative to the Root Object (i.e. the targetParent path name resolves to Device.Hosts.Host.). This is a strong reference (refType is strong), so the list items will be path names of valid object instances (and invalid items will be removed from the list).

```
<object name="Device.ManagementServer.ManageableDevice.{i}." ...>
  ...
  <parameter name="Host" access="readOnly">
    <description>{{list}} {{reference}} ...</description>
    <syntax>
      <list>
        <size maxLength="1024"/>
      </list>
      <string>
        <pathRef refType="strong" targetParent=".Hosts.Host." targetType="row"/>
      </string>
    </syntax>
  </parameter>
  ...
</object>
```

In the above example, the list has a max length of 1024 characters. This is a common practice with list-valued parameters of path references. If no max length were specified, then the default would have been 16 characters which would likely be insufficient. List-valued parameters are discussed in Section 6.6.5.

6.6.6.2 Instance-Reference Parameter

Note – The use of a path-reference parameter is preferred over an instance-reference parameter in referencing a table row. A path-reference parameter is defined to reference a table row by setting its target type to row.

An instance-reference parameter references an object instance (table row) via its instance number.

The reference is defined using an `instanceRef` facet element (I.13.2) within the referencing parameter's data type specification. Since the parameter value will be a specific instance number, the use of instanceRef only applies to int, unsignedInt and their derived types.

The `instanceRef/@targetParent` attribute specifies the path name of the multi-instance object (table) of which an instance (row) is to be referenced. This value cannot contain "{i}" placeholders since it must refer to a specific table. Note that it can still be within another table, but it must be possible to get there without crossing an "{i}" boundary.

The targetParent's path name scope can be specified using the optional `instanceRef/-@targetParentScope` attribute (with possible values: normal, model, object), however, this should be avoided when possible! The best practice is to use normal scope, the default if this attribute is omitted when first defining the parameter. Unfortunately there are some cases where you need to set the scope differently, e.g. where the path name begins with "Device.".

Note – Normal is a hybrid scope which usually gives the desired behavior (and where the scope of a path name is implied by the format of the path name itself). If the targetParent path begins with "Device" or "InternetGatewayDevice" then it is relative to the top of the naming hierarchy. If the targetParent path

© The Broadband Forum. All rights reserved

begins with a dot then it is relative to the Root or Service Object. Otherwise, the targetParent path is relative to the current object (and any leading hash characters in the targetParent path will redirect it up one level to the parent of the current object, or the parent's parent, and so on for each hash character). Path names and scope are discussed in more detail in Annex A.2.3.4/TR-106 [3].

The reference type, either strong or weak, is specified using the `instanceRef/@refType` attribute. Reference type is discussed in Section A.2.3.6/TR-106 [3]. In summary, a strong reference always either references a valid object instance or else is a null reference. A weak reference does not necessarily reference an existing object instance.

Note – A null reference indicates that a referencing 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. 0 for unsignedInt, -1 for int). See Section A.2.3.5/TR-106 [3].

The `targetParent` and `refType` attributes are both required for the `instanceRef` element.

The following example defines a reference parameter using an `instanceRef` facet. The reference parameter is named ParentID, and it references a row in the IKESA.{i} table. Within the ParentID parameter, the path name of the referenced table is specified via the `parameter/syntax/unsignedInt/instanceRef/@targetParent` attribute. Note that this is a relative path name, and since it is prefixed with a dot, it is relative to the Service Object (i.e. the targetParent path name resolves to FAPService.{i}.Transport.Tunnel.IKESA.). This is a strong reference (refType is strong), so the parameter value will either be a valid instance number or 0.

```
<object name="FAPService.{i}.Transport.Tunnel.IKESA.{i}." ...>
  ...
</object>

<object name="FAPService.{i}.Transport.Tunnel.ChildSA.{i}." ...>
  ...
  <parameter name="ParentID" access="readOnly">
    <description>{{reference}} ...</description>
    <syntax>
      <unsignedInt>
        <instanceRef refType="strong" targetParent=".Transport.Tunnel.IKESA."/>
      </unsignedInt>
    </syntax>
  </parameter>
  ...
</object>
```

6.6.6.3 Enumeration-Reference Parameter

An enumeration-reference parameter is basically an enumeration parameter which obtains its valid enumeration values from the runtime value of another parameter (or from a specified "null" value if the reference parameter is not itself a list). This other parameter is referenced by its path name and must be list-valued.

 © The Broadband Forum. All rights reserved

The reference is defined using an enumerationRef facet element (I.13.6) within the referencing parameter's data type specification. Since the parameter's value will be an enumeration value (a string), the use of enumerationRef only applies to string and its derived types.

The path name of the referenced parameter is specified using the `enumerationRef/-@targetParam` attribute. This is the only required attribute for the `enumerationRef` element. Its value cannot contain "{i}" placeholders since it has to to reference just one parameter. The referenced parameter must be a list-valued parameter.

The path name scope can be specified using the `enumerationRef/@targetParamScope` attribute (with possible values: normal, model, object), however, this should be avoided when possible! The best practice is to use normal scope, the default if this attribute is omitted when first defining the parameter. Unfortunately there are some cases where you need to set the scope differently, e.g. where the path name begins with "Device.".

Note – Normal is a hybrid scope which usually gives the desired behavior. If the targetParam path begins with "Device" or "InternetGatewayDevice" then it is relative to the top of the naming hierarchy. If the targetParam path begins with a dot then it is relative to the Root or Service Object. Otherwise, the targetParam path is relative to the current object (and any leading hash characters in the targetParam path will redirect it up one level to the parent of the current object, or the parent's parent, and so on for each hash character). Path names and scope are discussed in more detail in Annex A.2.3.4/TR-106 [3].

The following example defines a reference parameter using an `enumerationRef` facet. The reference parameter is named ModeEnabled (towards the bottom), and it references the list-valued parameter ModesSupported. Within the ModeEnabled parameter, the path name of the referenced parameter is specified via the `parameter/syntax/string/-enumerationRef/@targetParam` attribute. Note that this is a relative path name, and since it is prefixed with two hash characters, it is redirected to the Device.WiFi.EndPoint.{i} object two levels up in the naming hierarchy (i.e. the targetParam path name resolves to Device.WiFi.EndPoint.{i}.Security.ModesSupported).

```
<object name="Device.WiFi.EndPoint.{i}.Security." ...>
  ...
  <parameter name="ModesSupported" access="readOnly">
    <description>{{list}} ...</description>
    <syntax>
      <list/>
      <string>
        <enumeration value="None"/>
        <enumeration value="WEP-64"/>
        <enumeration value="WEP-128"/>
        ...
      </string>
    </syntax>
  </parameter>
  ...
</object>

<object name="Device.WiFi.EndPoint.{i}.Profile.{i}.Security." ...>
  ...
  <parameter name="ModeEnabled" access="readWrite">
    <description>{{reference}} ...</description>
    <syntax>
      <string>
        <enumerationRef targetParam="##.Security.ModesSupported"/>
```


© The Broadband Forum. All rights reserved

```
        </string>
      </syntax>
    </parameter>
    ...
  </object>
```

In the above example, the ModeEnabled parameter will always have a value that is one of the items from the ModesSupported parameter list. In some cases it is desirable to allow the referencing parameter to be set to some "null" value, indicating that none of the values of the referenced parameter currently apply. This is achieved using the `enumerationRef/-@nullValue` attribute.

6.6.7 Updating an Existing Parameter Definition

Sometimes an existing parameter needs to be updated. This is done using a `parameter` element (I.10) within a new revision of the document that defined the parameter in question. This implies that the parameter's other parent elements (e.g. `object`, `model` or `component`) will also be updated.

Note – The syntax for modifying a parameter is the same as for initially creating it, but there are rules. See Section A.2.10.1/TR-106 [3] for details.

An update to a parameter is a parameter definition that is based on the existing parameter. This is characterized by the use of the `parameter/@base` attribute, which indicates the name of the existing parameter being updated. Note that the `parameter/@name` attribute, which is used in the initial definition of a parameter, is not used when updating a parameter.

The updated definition will contain changes and additions to the existing parameter. It should not re-define those portions of the existing parameter that have not changed. Generally, this means that optional parameter attributes that are not changing are not re-specified (e.g. `status`, `activeNotify`), and that elements within the parameter that are not changing are also not re-specified (e.g. description, syntax). See Appendix I.10 for a full list of attributes and elements and whether they are optional or required.

Note – Required parameter attributes are always specified regardless of whether or not they are being changed, because if they were not then schema validation would fail. Optional parameter attributes, while generally not specified unless they are changing, may be re-specified if the author deems it appropriate.

Common reasons to update a parameter include:

- Change an attribute value; e.g. set status to deprecated
- Update its description
- Change its syntax; e.g. specify a default value, or narrow its set (or range) of valid values

The following example illustrates updates to two parameters (the first listing is the initial parameter definitions in tr-181-2-0-1 and the second listing is the updates in the much later tr-181-2-2-0). The Protocol parameter has its data type updated to indicate a maximum value, and the SourceUserClassID parameter just has its description updated. As expected, the updates are

 © The Broadband Forum. All rights reserved

within a new document revision and the parameters' parent elements are also updated (in this case, their object and model).

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-1">
  ...
  <model name="Device:2.0">
    ...
    <object name="Device.QoS.Classification.{i}." ...>
      ...

      <parameter name="Protocol" access="readWrite">
        <description>Classification criterion.
          Protocol number. A value of -1 indicates this criterion is not used for classification.
        </description>
        <syntax>
          <int>
            <range minInclusive="-1"/>
          </int>
          <default type="object" value="-1"/>
        </syntax>
      </parameter>

      <parameter name="SourceUserClassID" access="readWrite">
        <description>Classification criterion.
           A hexbinary string used to identify one or more LAN devices, value of the DHCP User
           Class Identifier (Option 77) as defined in...
        </description>
        <syntax>
          <hexBinary>
            <size maxLength="65535"/>
          </hexBinary>
          <default type="object" value=""/>
        </syntax>
      </parameter>

      ...
    </object>
    ...
  </model>
</dm:document>
```

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3" ...
             spec="urn:broadband-forum-org:tr-181-2-2-0">
  ...
  <model name="Device:2.2" base="Device:2.1">
    ...
    <object base="Device.QoS.Classification.{i}." ...>
      ...

      <parameter base="Protocol" access="readWrite">
        <syntax>
          <int>
            <range minInclusive="-1" maxInclusive="255"/>
          </int>
          <default type="object" value="-1"/>
        </syntax>
      </parameter>

      <parameter base="SourceUserClassID" access="readWrite">
        <description action="replace">Classification criterion.
           A hexbinary string used to identify one or more LAN devices, value of the DHCP User
           Class Identifier. The DHCP User Class Identifier is Option 77 (as defined in...
        </description>
      </parameter>

      ...
    </object>
```

 © The Broadband Forum. All rights reserved

```
    ...
  </model>
</dm:document>
```

A few things should be observed within the updated document (in the second listing) that applies whenever a parameter definition is updated:

- The `parameter/@base` attribute is used to reference the existing parameter definition.
- The parameters' required attributes are present (even though none were changed).
- The parameters' optional attributes are not present since none changed (though it is not an error to re-specify them). When omitted, the presumption is that their previous settings will carry forward unchanged.
- The Protocol parameter's description is omitted since it was not updated. No need to re-specify a description if it is not changing. When omitted, the presumption is that the existing description is carried forward.
- The SourceUserClassID parameter's syntax is omitted since it was not updated. No need to re-specify the syntax if it is not changing. When omitted, the presumption is that the existing syntax is carried forward.
- The Protocol parameter's updated data type definition is a restriction (not an extension) of its prior definition. Updates to a parameter's data type must obey the base type restriction rules of Section A.2.3.8/TR-106 [3] (i.e. a valid value of the updated data type definition will also be a valid value of its original data type definition; if a facet present in the base type is re-specified it must be fully specified; etc.).
- The Protocol parameter's default value is re-specified even though it was not updated. This is truly redundant (i.e. if omitted, the existing parameter default would be carried forward). However, it is not an error to re-specify unnecessary portions of the syntax, perhaps the author wished to provide some context for readability, but doing so could introduce errata.

Note – In the above example, it is not shown in the second listing, but the previous Device:2.1 model must be imported before it can be used to define the updated Device:2.2 model. See 6.4.2 for details on updating an existing model.

When an existing parameter needs be deprecated, obsoleted, or deleted, this is done by updating its `parameter/@status` attribute. The parameter's description must also be updated to include an explanation of why the status was changed. See Section 6.10 for details.

© The Broadband Forum. All rights reserved

6.7 Component

A component is a grouping of objects, parameters and/or profiles, which are defined within the component rather than directly within a Data Model. This means that they are not tied to any particular Root or Service model. They are defined at the top-level within a document, outside the scope of any models, using a `component` element (I.6).

Each component is assigned a unique name, by which it can be referenced, in order to later include its definition wherever it is needed. A component definition can be referenced (included) from within another component, model or object using a `component` reference element (I.7). Referencing (including) a component can be thought of as textual substitution (i.e. substituting the component reference element with the items defined within the actual component definition element).

In this way it is possible to define a set of objects, parameters and/or profiles that can be reused at different points within multiple Data Models. For example, TR-157 defines the DI_SupportedDataModel component which is included within all Root Data Models (i.e. within Device:2, Device:1, and InternetGatewayDevice:1). However, components can also be defined simply as a means to structure the definitions, even if reuse seems unnecessary.

Note that a component definition has no version number. Each time an existing component needs to be updated, it is actually re-defined using the same name.

Note – Objects, parameters, and profiles are defined and updated within a component in the same fashion that they are defined and updated within a model. For specific instruction on defining or updating these elements, see: object (Section 6.5), parameter (Section 6.6), and profile (Section 6.8).

6.7.1 Defining and Using a Component

The top-level `component` element (I.6) is used to define a new component. The component is assigned a name via its `component/@name` attribute. This name must be unique within the document (including imported components; Section 6.3.3). The use of this element is also discussed in Section A.2.5/TR-106 [3].

A component definition can contain objects, parameters, and/or profiles (i.e. in a similar fashion to how models contain such sub-elements). If `profile` elements are present, then they must appear at the end of the component after any `object` and `parameter` elements.

Each component definition should have a description. This is defined by the `component/description` element. The description (if present) must be the first element within the component.

The following example defines the component named UserInterface. It contains a description, objects and their parameters. It is defined within the document, but outside of the document's Device:2.0 model.

 © The Broadband Forum. All rights reserved

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-0">
  ...
  <import ...>
    ...
  </import>
  ...

  <component name="UserInterface">
    <description>Parameters related to the user interface of the CPE...</description>

    <object name="UserInterface." ...>
      ...
      <parameter name="PasswordReset" ...>
        <description>...</description>
        <syntax hidden="true">
          <boolean/>
        </syntax>
      </parameter>
      ...
    </object>
    ...
  </component>
  ...

  <model name="Device:2.0">
    ...
  </model>
</dm:document>
```

In the above example, note how the `component` definition element appears after any `import` elements and before any `model` element.

This example illustrates how to define a component, but does not indicate how this definition can be included within a Data Model (see below).

Referencing a component:

A component definition can be referenced (included) within another component, model, or object definition using the `component` reference element (I.7). This is a different type of component element, which uses its `component/@ref` attribute in order to reference a definition.

A component reference needs to specify the name of the component being included, and where to insert this component's items relative to the point of reference. The former is indicated by the `component/@ref` attribute (i.e. which component definition is being referenced). The latter is indicated by the `component/@path` attribute (i.e. the object path relative to the point of reference where the component's items will be inserted). The `path` attribute is optional; if omitted, then the component's items will be inserted exactly at the point of reference.

The following example builds on the preceding example. The UserInterface component is defined before the model, and it is referenced (included) within the model via a `model/component` reference element[22]. The `model/component/@ref` attribute specifies

[22] Broadband Forum standard Data Models no longer define both components and models within the same DM Instance. The information has been retained here in case vendors wish to do something similar in their vendor-specific Data Models.

 © The Broadband Forum. All rights reserved

the UserInterface component. The `model/component/@path` attribute specifies a path of "Device.". This means, for example, that the component's items, when included, resolve to Device.UserInterface (an included object) and Device.UserInterface.PasswordReset (an included parameter).

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-0">
  ...

  <component name="UserInterface">
    ...
  </component>
  ...

  <model name="Device:2.0">
    ...
    <object name="Device." ...>
      ...
    </object>
    ...
    <component path="Device." ref="UserInterface"/>
    ...
  </model>
</dm:document>
```

In the above example, the component reference element was placed directly within the model element (referenced from within the model definition), and a `model/component/@path` attribute indicated that the relative path for the included items should actually be "Device.". An equivalent alternative could have just as easily been to place the component reference element directly within the Device object and to then omit the `path` attribute (meaning that the relative path of the included items is at the point of reference). For example:

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-0">
  ...

  <component name="UserInterface">
    ...
  </component>
  ...

  <model name="Device:2.0">
    ...
    <object name="Device." ...>
      ...
      <component ref="UserInterface"/>
      ...
    </object>
    ...
  </model>
</dm:document>
```

However, the preference is to place component reference elements directly within the model element (not within an object element). This mirrors how objects are defined within a model (all at the top-level). A notable exception might be when a component definition contains top-level parameters and no objects.

 © The Broadband Forum. All rights reserved

6.7.2 Importing and Using a Component

Note – This section builds on Section 6.7.1, especially its discussion on referencing a component.

Since a component is not tied to any particular Root or Service model, it can be defined in one document, and then imported and included into a model (or component or object) defined in another document. A component definition is imported using an `import/component` element (see Sections 6.3 and 6.3.3), and is then included (referenced) via a `component` reference element (I.7).

The following example imports the NSLookupDiag component definition from tr-157-1-2.xml, using an import/component element. This makes the remote component definition visible (to be referenced) within the local document. It is referenced (included) within the local document's model via a model/component reference element.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-0">
  ...
  <import file="tr-157-1-2.xml" spec="urn:broadband-forum-org:tr-157-1-2">
    ...
    <component name="NSLookupDiag"/>
    ...
  </import>
  ...

  <model name="Device:2.0">
    ...
    <object name="Device.DNS.Diagnostics." ...>
      ...
    </object>
    ...
    <component path="Device.DNS.Diagnostics." ref="NSLookupDiag"/>
    ...
  </model>
</dm:document>
```

In the above example, the `model/component/@ref` attribute specifies the NSLookupDiag component that was imported. The `model/component/@path` attribute specifies a relative path of "Device.DNS.Diagnostics.". This means, for example, that the imported component's items (e.g. its NSLookupDiagnostics object and its NSLookupDiagnostics.Interface parameter), when included, resolve to Device.DNS.Diagnostics.NSLookupDiagnostics (an included object) and Device.DNS.Diagnostics.NSLookupDiagnostics.Interface (an included parameter).

6.7.3 Updating an Existing Component

Note – This section builds on concepts discussed in Section 6.7.1.

A component definition does not have a version number. Each time an existing component needs to be updated, a new component with the same name (based on the previous component definition) is defined in a new version of the document. Generally the new version of the document is an amendment.

At a minimum, the amended document will need to:

© The Broadband Forum. All rights reserved

- Import the previous version of the component. This is done using an `import/-component` element (see Section 6.3.3). Note that the imported component is referred to locally by a different name (usually the local name is the same as the remote name but is prefixed with an underscore).
- If the amended document also needs to update a model, then import the previous version of the model using an `import/model` element (see Section 6.3.2). ***Note**: Broadband Forum standard Data Models no longer define both components and models within the same DM Instance.*
- Define a new "diffs" component that will only contain the changes needed to extend the imported component. This is done using a `component` definition element (I.6), discussed in Section 6.7.1. The convention is to name this component as the concatenation of the imported component name and the text "Diffs". The name should <u>not</u> be prefixed with an underscore. This is because names that begin with underscores are local and cannot be imported by other files; but the Diffs component has to be importable.
- Redefine the original component (using a `component` definition element; I.6) as a union of the imported component and the "diffs" component (using `component` reference elements; I.7). This is actually a new component defined within the local document, which has the same name as the remotely defined component.
- If the amended document needs to update a model, then reference (include) the "diffs" component from within the updated model using the `model/component` element (I.7). ***Note**: Broadband Forum standard Data Models no longer define both components and models within the same DM Instance.*

The following example demonstrates this process, where the imported component is defined in the previous version of the same document. This example builds on the UserInterface example from Section 6.7.1.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3" ...
             spec="urn:broadband-forum-org:tr-181-2-2-0">
  ...
  <import file="tr-181-2-1.xml" spec="urn:broadband-forum-org:tr-181-2-1">
    ...
    <component name="_UserInterface" ref="UserInterface"/>
    <model name="Device:2.1"/>
  </import>

  <component name="UserInterfaceDiffs">
    <object base="UserInterface." access="readOnly" minEntries="1" maxEntries="1">
      <parameter base="PasswordReset" access="readWrite">
        <syntax command="true">
          <boolean/>
        </syntax>
      </parameter>
    </object>
  </component>

  <component name="UserInterface">
    <component ref="_UserInterface"/>
    <component ref="UserInterfaceDiffs"/>
  </component>

  <model name="Device:2.2" base="Device:2.1">
    <component path="Device." ref="UserInterfaceDiffs"/>
    ...
```


© The Broadband Forum. All rights reserved

```
  </model>
</dm:document>
```

In the above example, the imported component is defined remotely as UserInterface and is referred to locally as _UserInterface. The updates needed to extend the imported component are defined solely within the new UserInterfaceDiffs component. The UserInterface component is then redefined within the local document as the union of the _UserInterface and UserInterfaceDiffs components. This means that the new UserInterface component contains the full definition (i.e. previous version plus changes).

Note that the above example also revises the associated model (defines Device:2.2 based on the previous Device:2.1), in order to incorporate the component updates within the model. Since the revised model only needs the changes, the model/component element simply references the "diffs" component.

Separate documents:

It is also possible and recommended that the component to be updated, and the model in which it is included, are defined in different documents. The basic process in updating and including the component is the same as discussed above, but the work is split across both documents.

The following listings demonstrate this process; the updated component is defined in tr-157-1-3 and the updated model is defined in tr-181-2-1[23]. This example builds on the NSLookupDiag example from Section 6.7.2.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3" ...
             spec="urn:broadband-forum-org:tr-157-1-3-0">
  ...
  <import file="tr-157-1-2.xml" spec="urn:broadband-forum-org:tr-157-1-2">
    ...
    <component name="_NSLookupDiag" ref="NSLookupDiag"/>
    ...
  </import>
  ...
  <component name="NSLookupDiagDiffs">
    <object base="NSLookupDiagnostics.Result.{i}." ...>
      <parameter base="Status" ...>
        <description action="replace">
          Result Parameter representing whether NS Lookup was successful...
        </description>
      </parameter>
    </object>
  </component>

  <component name="NSLookupDiag">
    <component ref="_NSLookupDiag"/>
    <component ref="NSLookupDiagDiffs"/>
  </component>
  ...
</dm:document>
```

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3" ...
             spec="urn:broadband-forum-org:tr-181-2-1-0">
  ...
```

[23] Note that tr-157-1-3 defines its own models, which also reference NSLookupDiagDiffs, but these have been omitted from this example since the concept is already covered in the previous example.

 © The Broadband Forum. All rights reserved

```
  <import file="tr-157-1-3.xml" spec="urn:broadband-forum-org:tr-157-1-3">
    ...
    <component name="NSLookupDiagDiffs"/>
    ...
  </import>
  ...

  <model name="Device:2.1" base="Device:2.0">
    ...
    <component path="Device.DNS.Diagnostics." ref="NSLookupDiagDiffs"/>
    ...
  </model>
</dm:document>
```

In the above tr-157-1-3 document listing, the NSLookupDiag component import, the definition of the “diffs” component, and the redefinition of the imported component are all specified in the same fashion as described in the previous example. However, the tr-181-2-1 document updates are specified somewhat differently.

The tr-181-2-1 document imports the NSLookupDiagDiffs component from the tr-157-1-3 document. Since this component is not updated within the tr-181-2-1 document (i.e. read-only), it is imported using the `import/component/@name` attribute alone (i.e. omits the `ref` attribute; this means that the NSLookupDiagDiffs component is referred to locally by the same name that it is defined by in the remote document). Finally, the `model/component` element references the NSLookupDiagDiffs component in order to include the component updates within the revised model.

 © The Broadband Forum. All rights reserved

6.8 Profile

A profile is a named group of requirements associated with a model or component. It is specified using the `profile` element (I.11). Profile usage is also discussed in Section 2.3/TR-106 [3].

Each profile element has a name specified using the `profile/@name` attribute, the `profile/@base` attribute, or both (depending on whether it is a new profile, it updates an existing profile, or it extends an existing profile, respectively). A profile's name is formatted as follows: the name, a colon, and its version number (e.g. the initial version of a Baseline profile is named Baseline:1). Profile versioning is discussed in Section 2.3.3/TR-106 [3].

Each profile name is unique within the major version of its associated model.

A profile contains references to this model's objects and/or parameters. These references are specified using the object (I.11.1) and parameter (I.11.2) reference elements. Their `ref` attribute indicates the name of the object[24] or parameter being referenced, and their `requirement` attribute indicates the access requirement for that item.

A profile object requirement can be one of: notSpecified, present, create, delete, or createDelete. This is set using a `profile/object/@requirement` attribute. The create, delete, and createDelete settings only apply to multi-instance objects.

A profile parameter requirement can be one of: readOnly or readWrite. This is set using a `profile/parameter/@requirement` attribute.

A profile can have a description. This is defined using the profile/description element. Similarly, a profile's objects and parameters can also have descriptions. However, most profiles (and their objects and parameters) do not have a description unless there are additional normative requirements that need to be specified.

Note – The examples throughout this section are of profiles defined within models. The mechanism is almost identical in defining profiles within components. However, a component's profile name must be unique not only within the component itself, but also within (the major version of) any models from which the component is referenced (included).

6.8.1 Defining a New Profile

The initial version of a new profile is defined using a `profile/@name` attribute alone (i.e. omit the profile's base and extends attributes). This attribute indicates the name and version of the profile. The version of a new profile always starts at 1.

[24] For model profiles, `model/profile/object/@ref` will be the full path name of the referenced object. For component profiles, `component/profile/object/@ref` will be a path name that is relative to the component's top-level object.

© The Broadband Forum. All rights reserved

The profile will contain references to objects and/or parameters, where each item indicates its access requirement. Objects are referenced using their path name.

The following example defines the Baseline:1 profile within the Device:2 model. Note that the profile is defined at the bottom of the model, after any object definitions and component references.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-0">
  ...

  <model name="Device:2.0">
    ...
    <object name="Device." ...>
      ...
    </object>
    ...

    <profile name="Baseline:1">
      <object ref="Device." requirement="present">
        <parameter ref="InterfaceStackNumberOfEntries" requirement="readOnly"/>
      </object>
      <object ref="Device.DeviceInfo." requirement="present">
        <parameter ref="Manufacturer" requirement="readOnly"/>
        <parameter ref="ProvisioningCode" requirement="readWrite"/>
        ...
      </object>
      ...
    </profile>
    ...
  </model>
</dm:document>
```

Note – A best practice is to define a Baseline profile for each Root and Service Data Model. The Baseline profile should define a minimum set of object and parameter requirements for a CPE implementing the associated model. See Section 2.3.4/TR-106 [3].

6.8.2 Updating an Existing Profile

An updated version of an existing profile is defined using both a `profile/@name` and a `profile/@base` attribute.

Note – The syntax for modifying a profile is similar as for initially creating it, but there are rules. These rules are not specified in the DM Schema. See Section A.2.10.3/TR-106 [3] for details.

The `base` attribute indicates the profile name (and version number) of the existing profile, and the `name` attribute indicates this same name but with an incremented version number for the new (updated) profile. The version of an updated profile is always incremented by 1.

The updated profile will only contain references to those objects and/or parameters whose requirement is changing (i.e. adding or removing an item, or changing an existing item's requirement).

 © The Broadband Forum. All rights reserved

The following example defines the Baseline:2 profile based on the existing Baseline:1 profile. The Baseline:1 profile is visible (can be referenced) within the local document because the Device:2.1 model that defines it has been imported via the import/model element.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3" ...
             spec="urn:broadband-forum-org:tr-181-2-2-0">
  ...

  <import file="tr-181-2-1.xml" spec="urn:broadband-forum-org:tr-181-2-1">
    ...
    <model name="Device:2.1"/>
  </import>
  ...

  <model name="Device:2.2">
    ...
    <object ...>
      ...
    </object>
    ...

    <profile name="Baseline:2" base="Baseline:1">
      <object ref="Device.InterfaceStack.{i}." requirement="present">
        <parameter ref="HigherLayer" requirement="readOnly" />
        <parameter ref="LowerLayer" requirement="readOnly" />
      </object>
      <object ref="Device.DNS." requirement="present">
        <parameter ref="SupportedRecordTypes" requirement="readOnly"/>
      </object>
    </profile>
    ...
  </model>
</dm:document>
```

In the above example, the total object and parameter requirements for Baseline:2 are actually the union of those from Baseline:1 plus the new (or updated) requirements specified in Baseline:2.

When an existing profile needs be deprecated, obsoleted, or deleted, this is done by updating its `profile/@status` attribute. See Section 6.10 for details. Note that such a status update will also apply to the profile's contained items (e.g. object and parameter references), but only if this will "promote" a given item's status to a "higher" value.

6.8.3 Defining a New Profile by Extension

The initial version of a new profile, which is an extension of (inherits from) another profile, is defined using both a `profile/@name` and a `profile/@extends` attribute.

The name attribute indicates the name and version of the new profile. The version of a new profile always starts at 1.

The `extends` attribute indicates the name and version of the profile(s) being extended (inherited from). This attribute can be a list, and so inheritance from multiple profiles is supported.

 © The Broadband Forum. All rights reserved

The following example defines the DHCPv6ServerAdv:1 profile, which specifies the `profile/@extends` attribute, and so inherits requirements from the DHCPv6Server:1 profile.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3" ...
             spec="urn:broadband-forum-org:tr-181-2-2-0">
  ...

  <model name="Device:2.2">
    ...
    <object ...>
      ...
    </object>
    ...

    <profile name="DHCPv6Server:1">
      <object ref="Device.DHCPv6." requirement="present"/>
      <object ref="Device.DHCPv6.Server." requirement="present">
        <parameter ref="Enable" requirement="readWrite"/>
        <parameter ref="PoolNumberOfEntries" requirement="readOnly"/>
      </object>
      ...
    </profile>

    <profile name="DHCPv6ServerAdv:1" extends="DHCPv6Server:1">
      <object ref="Device.DHCPv6.Server.Pool.{i}." requirement="createDelete">
        <parameter ref="DUID" requirement="readWrite"/>
        <parameter ref="VendorClassID" requirement="readWrite"/>
        <parameter ref="UserClassID" requirement="readWrite"/>
        <parameter ref="SourceAddress" requirement="readWrite"/>
        <parameter ref="SourceAddressMask" requirement="readWrite"/>
      </object>
    </profile>
    ...
  </model>
</dm:document>
```

In the above example, the total object and parameter requirements for DHCPv6ServerAdv:1 are actually the union of those from DHCPv6Server:1 (via inheritance) plus the requirements specified in DHCPv6ServerAdv:1.

© The Broadband Forum. All rights reserved

6.9 Description

A description is free text which can contain a limited amount of MediaWiki-like markup (I.2.2) that processing tools can interpret.

Many different types of elements (e.g. an `object` element) can contain a description. A description is specified using a `description` element (I.2) within the element to be described (e.g. an `object/description` element describes its object parent). The description must be the first element to appear within the parent element.

Descriptions can be used to explain the purpose and intent of the parent element. For example, how it works, what it is used for, how it relates to other elements within the Data Model, additional normative requirements, citing bibliographic references, etc.

To determine whether a particular element can have a description, see its reference section in Appendix I. In practice, the following elements most often contain a description:

- `document` (I.1)
- `dataType` definition (I.4)
- `component` definition (I.6)
- `object` definition (I.9)
- `parameter` definition (I.10)
- `enumeration` and `pattern` facets (I.13)
- `profile` (I.11) and its `object` and `parameter` references (I.11.1 and I.11.2 respectively)

Note – The preference is to limit profile descriptions (and `profile/object` and `profile/parameter` descriptions) to any additional requirements that cannot be expressed via the element attributes; i.e. not used for general descriptive purposes as can be the case with other parent elements such as object and parameter.

Note – For Broadband Forum documents, the character set must be restricted to printable characters in the Basic Latin Unicode block. Effectively, this means "printable ASCII", including line feed. Descriptions are also discussed in Section A.2.2/TR-106 [3].

6.9.1 Defining a Description

A description is defined using the `description` element (I.2). It can appear within many different parent elements (e.g. `object`, `parameter`). Refer to Appendix I to determine whether a particular element can have a description.

The description element has one attribute (`description/@action`) which indicates how the description should be processed. For a new description definition, the `action` attribute can only be set to "create" (the default). The preference is to always omit this attribute when defining a description for the first time.

© The Broadband Forum. All rights reserved

The following example illustrates an object description. It is simply free text within a description element. As required, the description is the first element within its parent element.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-0">
  ...
  <model name="Device:2.0">
    ...
    <object name="Device.IP.Interface.{i}." ...>
      <description>IP interface table (a stackable interface object as described in {{bibref|TR-
181i2|Section 4.2}}). This table models the layer 3 IP interface.</description>
      ...
    </object>
    ...
  </model>
</dm:document>
```

Note – The above description example includes a `{{bibref}}` Template (markup) used to reference a document within the bibliography. See Section 6.1.3 for a discussion on citing bibliographic references.

The following example expands on the previous to illustrate the various elements that will most often have a description. This includes (but is not limited to): the document, component definitions and their objects and parameters, a model's objects and parameters, occasionally parameter enumerations, and (for the purpose of specifying additional requirements) profiles and their object and parameter references.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-2" ...
             spec="urn:broadband-forum-org:tr-181-2-0-0">
  <description>Device:2 Data Model.</description>
  ...

  <component name="DeviceInfo">
    <description>General information about the device...</description>

    <object name="DeviceInfo." ...>
      <description>This object contains general device information.</description>

      <parameter name="Manufacturer" ...>
        <description>The manufacturer of the CPE (human readable string).</description>
        ...
      </parameter>
      ...
    </object>
    ...
  </component>

  <model name="Device:2.0">
    ...
    <object name="Device.IP.Interface.{i}." ...>
      <description>IP interface table (a stackable interface object as described in {{bibref|TR-
181i2|Section 4.2}}). ...</description>
      ...

      <parameter name="Enable" ...>
        <description>Enables or disables the interface. ...</description>
        ...
      </parameter>
      ...
    </object>
    ...

    <object name="Device.ATM.Link.{i}." ...>
      <description>ATM link-layer table...</description>
      ...
```

 © The Broadband Forum. All rights reserved

```
      <parameter name="LinkType" ...>
        <description>Indicates the type of connection ...</description>
        <syntax>
          <string>
            <enumeration value="EoA">
              <description>{{bibref|RFC2684}} bridged Ethernet over ATM</description>
            </enumeration>
            <enumeration value="IPoA">
              <description>{{bibref|RFC2684}} routed IP over ATM</description>
            </enumeration>
            ...
          </string>
        </syntax>
      </parameter>
      ...
    </object>
    ...

    <profile name="DeviceAssociation:1">
      <description>This profile implies support for all of the Gateway requirements defined in
{{bibref|TR-069|Annex F}}.</description>
      ...
    </profile>
    ...

    <profile name="DHCPv4Client:1">
      ...
      <object ref="Device.DHCPv4.Client.{i}.SentOption.{i}." ...>
        <description>This table is REQUIRED to support sending of option 60 ...</description>
        ...
      </object>
    </profile>
    ...

  </model>
</dm:document>
```

6.9.2 Updating an Existing Description

Sometimes an existing description needs to be updated. This will be done using a `description` element (I.2) within a new revision of the document that defined the description in question. This implies that the description's other parent elements (e.g. `parameter`, `object`, `model`) will also be updated.

The text provided in the `description` element will be used to update the existing description. The `description/@action` attribute indicates how the new text relates to the existing description. This attribute can have one of the following values: prefix, append, replace.

If the action is "prefix" then the updated description will be the combination of the new text followed by the existing description. If the action is "append" then the updated description will be the combination of the existing description followed by the new text. If the action is "replace" then the new text is an outright replacement of the original text.

Note that prefixed and appended text is always regarded as separate paragraphs from the description text previously defined.

The following example illustrates a description update for the IP Interface object and its Enable parameter from the previous example.

 © The Broadband Forum. All rights reserved

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-3" ...
             spec="urn:broadband-forum-org:tr-181-2-2-0">
  ...
  <model name="Device:2.2" base="Device:2.1">
    ...

    <object base="Device.IP.Interface.{i}." ...>
      <description action="append">Each IP interface can be attached to the IPv4 and/or IPv6
stack. ...</description>

      <parameter base="Enable" ...>
        <description action="replace">Enables or disables the interface (regardless of
{{param|IPv4Enable}} and {{param|IPv6Enable}}). ...</description>
      </parameter>
      ...
    </object>
    ...

  </model>
</dm:document>
```

In the above example, the object description indicates an append action. This means that the new text will be appended to the end of the existing description for the object, to give something like: "IP interface table… Each IP interface can be attached to the IPv4 and/or IPv6 stack. …". The parameter description indicates a replace action. This means that the new text will completely replace any previous description for the parameter.

6.9.3 Laying Out Descriptions

How a description layout is interpreted by processing tools (such as the Report Tool, e.g. when generating an HTML report) is affected by whitespace pre-processing rules and by MediaWiki-like markup present within the `description` element.

6.9.3.1 Whitespace Pre-processing

A description's leading whitespace (up to and including the first line break) and trailing whitespace (including line breaks) is not significant. Processing tools need to discard non-significant whitespace. This will allow for a variety of layout styles while still retaining predictable behavior. Pre-processing rules are detailed in Section I.2.1.

The following example illustrates several single-line description fragments that would be rendered equivalent by the whitespace pre-processing rules. The first style shown is preferred when writing XML.

```
      <description>The top-level object for a Device</description>

      <description>   The top-level object for a Device   </description>

      <description>The top-level object for a Device
</description>

      <description>The top-level object for a Device
      </description>
```

 © The Broadband Forum. All rights reserved

Non-significant whitespace also includes the longest common whitespace prefix that occurs at the start of every line. For example, if every line in a description is indented by three whitespace characters then these characters are not part of the description layout. However, if one of these lines had no whitespace indent, then the three whitespace characters at the beginning of each of the other lines would be significant.

Note that this common-prefix calculation is based on the number of whitespace characters, not the type of whitespace character (i.e. the space character and the tab character are not differentiated). For this reason, a description should not contain tab characters; otherwise the rendered layout will not be consistent.

The following example illustrates some additional single-line description fragments that would be rendered equivalent by the whitespace pre-processing rules. Each of the styles shown is reasonable, but the first one might be slightly preferred when writing XML (especially for short descriptions).

```
      <description>The top-level object for a Device</description>

      <description>
         The top-level object for a Device
      </description>

      <description>
           The top-level object for a Device
      </description>
```

The following example illustrates some multi-line description fragments that would be rendered equivalent by the whitespace pre-processing rules. The first or second style shown is preferred when writing XML.

```
      <description>Numeric value indicating the supported revision for UPnP IGD.
A value of 0 indicates no support.</description>

      <description>
         Numeric value indicating the supported revision for UPnP IGD.
         A value of 0 indicates no support.
      </description>

      <description>
      Numeric value indicating the supported revision for UPnP IGD.
      A value of 0 indicates no support.
      </description>

      <description>
Numeric value indicating the supported revision for UPnP IGD.
A value of 0 indicates no support.
      </description>

      <description>Numeric value indicating the supported revision for UPnP IGD.
A value of 0 indicates no support.
      </description>
```

6.9.3.2 Markup

A description can contain a limited amount of markup that is similar in fashion to MediaWiki markup. Processing tools are expected to interpret this markup when processing descriptions (e.g. the Report Tool interprets the markup when generating its HTML Data Model reports).

 © The Broadband Forum. All rights reserved

There are two types of markup available: formatting markup (I.2.2) which indicates how the marked text should be formatted, and Template references (I.2.3) which are meant to be replaced with template-dependent text.

Formatting markup is indicated in a description using certain reserved character sequences. For example, two apostrophes on each side of some text indicate that the contained text is to be emphasized in italics (e.g. `''`This is to be italicized`''`). Possible formatting includes: italics, bold, bold italics, bulleted list, numbered list, indented list, verbatim. See Section I.2.2 for a full list of the standard markup and how it is invoked.

A Template reference is indicated in a description as text enclosed in double curly braces ({}). The template text consists of a template name that can be followed by arguments separated by vertical pipe (|) characters. For example, `{{bibref|TR-106a6|A.2.2.4}}` is a bibref Template, and would be replaced with something like [Section A.2.2.4/TR-106a6]. See Section I.2.3 for a full list of the standard Templates and how they are invoked.

The following example illustrates a description with four lines containing various markup items. A processing tool should interpret the markup as follows: `{{bibref|RFC3986}}` is a Template reference to an item defined in the bibliography; the line starting with a colon indicates an indented line; the `''host''` text is to be italicized; the URL literal is to be a hyperlink; and each line is a separate paragraph. Also note that whitespace pre-processing is applied before any markup is interpreted.

```
      <description>
         HTTP URL, as defined in {{bibref|RFC3986}}.
         In the form:
         : http://host:port/path
         The ''host'' portion of the URL MAY be the IP address for the management interface of
the CPE in lieu of a host name.
      </description>
```

The resulting layout will look something like the following:

```
HTTP URL, as defined in [RFC3986].

In the form:
   http://host:port/path

The host portion of the URL MAY be the IP address for the management interface of the CPE in lieu
of a host name.
```


© The Broadband Forum. All rights reserved

6.10 Status: Deprecate, Obsolete, Delete

Many different types of elements (e.g. `parameter`) can be deprecated, obsoleted, or deleted. This is specified using an element's `status` attribute (e.g. `parameter/@status`). See Section 2.4/TR-106 [3] for requirements on deprecated and obsoleted items.

The default value when first defining a `status` attribute is "current", meaning its element is currently available in the document. The convention is to omit the `status` attribute when initially defining an element (in favor of its default value). Therefore, the `status` attribute goes unnoticed until the need arises to deprecate, obsolete, or delete an element.

When an element's status is updated it must be a "promotion" 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.

To determine whether a particular element supports a `status` attribute, see its reference section in Appendix I. In practice, the following elements most often utilize status:

- `object` definition (I.9)
- `parameter` definition (I.10)
- `enumeration` and `pattern` facets (I.13)
- `profile` (I.11) and its `object` and `parameter` references (I.11.1 and I.11.2 respectively)

However, many elements support the `status` attribute in order to handle the need for error correction. For example, if a parameter is defined with a `default` value or a `size` range element that turns out to be unwarranted, it can be fixed by deleting the offending element (i.e. set its status to deleted) while leaving the remainder of the parameter definition intact.

The following example uses the `status` attribute to obsolete the QueueKey parameter and to delete the ShapingBurstSize parameter's default element.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-0" ...
             spec="urn:broadband-forum-org:tr-098-1-1-0">
  ...
  <model name="InternetGatewayDevice:1.2" base="InternetGatewayDevice:1.1">
    ...

    <object base="InternetGatewayDevice.QueueManagement.Queue.{i}." ...>
      ...
      <parameter base="QueueKey" access="readOnly" status="obsoleted">
        <description action="append">
          This parameter is OBSOLETED because it serves no purpose (no parameter references it).
        </description>
      </parameter>

      <parameter base="ShapingBurstSize" access="readWrite">
        ...
        <syntax>
          <unsignedInt/>
          <default type="object" value="0" status="deleted"/>
        </syntax>
```

 © The Broadband Forum. All rights reserved

```
      </parameter>
    ...
    </object>

    ...
  </model>
</dm:document>
```

Note that whenever an element is deprecated or obsoleted, it needs a description that explains why the status change occurred. This is shown with the obsoleted QueueKey parameter above.

 © The Broadband Forum. All rights reserved

7 DT XML Data Model Tutorials

This section provides guidance in defining DT Instance documents (called DT Instances for short). These documents are XML files that comply with the DT Schema (Appendix II). They are defined by CPE vendors in order to specify a supported Data Model for a given device type, where a supported Data Model is defined in terms of one or more DM Data Models (i.e. specifying the portions of standard, or vendor-extended, Root and/or Service Data Models that are actually implemented by a particular device type).

Note that the Broadband Forum does not publish DT Instance documents. Rather, DT Instance development is the realm of CPE hardware and software vendors.

A CPE vendor can specify a device type's supported Data Model using one or more DT Instances, that each reference an associated DM Data Model. This means that a device type can support multiple DM Data Models (i.e. multiple DT Instances where some reference a different DM Data Model). But it also means that there can be multiple DT Instances that reference the same DM Data Model, e.g. there could be a DT Instance that just covers the DHCPv6 client Data Model, or the UPnP Discovery Data Model. Often there will be a DT Instance for an installable/uninstallable software module if it adds additional Data Model support (this keeps the DT Instance specifications modular).

This modular approach allows a CPE vendor to represent a supported Data Model as a set of documents where each document represents a portion of the entire Data Model.

If some DT definition (e.g. a supported parameter syntax) conflicts with its corresponding DM definition, the DT definition is clipped to fall within what is allowed by the DM (i.e. if there is a conflict then the DM definition wins).

Note – Each DT Instance is associated with a particular device type (and multiple DT Instances can be of the same device type). This is indicated by the document element's `deviceType` attribute (i.e. `document/@deviceType`). The device type is represented as a globally unique URI. The owner of this URI can either be the CPE hardware or software vendor.

Note – A CPE can publish its supported Data Model via its DeviceInfo.SupportedDataModel.{i} table. Each table entry represents a different DT Instance, but taken together the table represents the CPE's entire supported Data Model. See TR-157 [5] for details.

 © The Broadband Forum. All rights reserved

7.1 Bibliography

Note – Bibliographic references are defined in the DT Schema in the same way that they are defined in the DM Schema (6.1). They are included in the DT Schema for completeness, but it is expected that they will rarely be used. This is because bibliographic citations would likely be made from within a DM Instance document rather than within a DT Instance document.

DT bibliography references are defined using the `document/bibliography/reference` element (II.5.1), and can be cited from within an `annotation` element (II.2), e.g. from within object and parameter annotations, using a `{{bibref}}` Template (I.2.3).

This is analogous to the DM bibliography; see Section 6.1 for further details in defining and citing bibliographic references[25].

In the following example, bibliography `reference` RFC2717 is defined within the DT Instance. Then it is cited from the ContentReferenceId parameter's `annotation` using the Template notation `{{bibref|RFC2717}}`.

```
<dt:document xmlns:dt="urn:broadband-forum-org:cwmp:devicetype-1-1" ...
             deviceType="urn:your-company-com:some-device-type-id">
  ...
  <import file="tr-135-1-1.xml" spec="urn:broadband-forum-org:tr-135-1-1">
    <model name="STBService:1.1"/>
  </import>
  ...
  <bibliography>
    <reference id="RFC2717">
      <name>RFC 2717</name>
      <title>Registration Procedures for URL Scheme Names</title>
      <organization>IETF</organization>
      <category>RFC</category>
      <hyperlink>http://www.ietf.org/rfc/rfc2717.txt</hyperlink>
    </reference>
    ...
  <bibliography>
  ...
  <model...>
    <object ...>
      ...
      <parameter ref="ContentReferenceId" access="readOnly">
        <annotation>Some additional comment about {{bibref|RFC2717}}...</annotation>
        ...
      </parameter>
    </object>
    ...
  </model>
</dt:document>
```

The `bibliography` element (if present) comes after the `annotation` element and `import` elements, and before other top-level elements such as `feature` and `model`.

[25] In a DT Instance, bibliographic references are cited from within `annotation` elements. This differs from DM Instances, which cite bibliographic references from within `description` elements.

 © The Broadband Forum. All rights reserved

7.2 Import

A DT Instance can import items from a DM Instance via an `import` element (II.4). By doing so, these imported items are made visible within the DT Instance.

This is analogous to the DM import (6.3), except that a DT Instance cannot import components[26]. The following items can be imported from DM Instance documents:

- Bibliography (an entire bibliography; see Section 6.3 for details)
- Named data types
- Models

Each `import` element will specify the file and spec of the DM Instance from which items are being imported. No more than one `import` element is specified for a particular DM Instance document.

`Import` elements come after the `annotation` element (if present), and before other top-level elements such as `bibliography`, `feature`, and `model`.

7.2.1 Import a Named Data Type

Note – The file tr-106-1-0-0-types.xml is the centralized document that contains named data type definitions. Generally, named data types will be defined there and be imported into DT Instance documents as needed. Named data types can also be defined in other DM Instance documents, but this is not the norm.

A named data type is imported into a DT Instance using the `import` element (II.4), which indicates the DM Instance file to import from, and the `import/dataType` element, which indicates the specific data type definition to import. The `dataType` element is repeated for each named data type to be imported.

Note that once a named data type has been imported into a DT Instance, it is visible within that document, and can be used for example in specifying how the device type will support syntax for model parameters.

The following example shows the IPAddress and MACAddress named data types being imported (from tr-106-1-0-types.xml) into a DT Instance document.

```
<dt:document xmlns:dt="urn:broadband-forum-org:cwmp:devicetype-1-1" ...
             deviceType="urn:your-company-com:some-device-type-id">
  <annotation>...</annotation>

  <import file="tr-106-1-0-types.xml" spec="urn:broadband-forum-org:tr-106-1-0">
    <dataType name="IPAddress"/>
    <dataType name="MACAddress"/>
  </import>
  ...
</dt:document>
```

[26] A DT Instance does not need to import components, because they will already be part of a DM model definition.

 © The Broadband Forum. All rights reserved

The `import/@file` attribute indicates the name of the import file. The `import/@spec` attribute (optional) indicates the spec value of the import file. The `import/dataType/-@name` attribute indicates the name of the data type to be imported from the import file.

7.2.2 Import a Data Model

A DM Data Model is imported into a DT Instance using the `import` element (II.4), which indicates the DM Instance file to import from, and the `import/model` element, which indicates the specific model:version to import.

Note – A DT Instance must only relate to a single DM Data Model (a single Root object or Service object). This means that only one DM model may be imported into a DT Instance document. This requirement is specified by TR-157 [5], in the DeviceInfo.SupportedDataModel.{i} table description.

For a given DM Data Model, only the highest revision supported by the device type need be imported. For example, if a device type supports Device:2.2 then only this revision of the Device:2 Data Model is imported. Since the Device:2.2 Data Model is actually the aggregate of all its previous revisions, the DT Instance document will have imported the entire Device:2 model definition up to and including the Device:2.2 revision.

Note – If a CPE implements a vendor-specific extension to a standard DM Data Model, then the expectation is that there would be two DT Instances: one that covers the vendor-specific DM model and another that covers the standard DM model. In other words, there are separate DT Instances for standard and vendor-specific DM models[27].

Once a DM Data Model has been imported into a local document, it is visible within that document, and can be used for example in specifying how the device type will support this Data Model.

In the following example the Device:2.2 Data Model is imported into the DT Instance, via the `import/model` element. It imports Device:2.2 in order to then reference this Data Model via the top-level model element. The implication is that later revisions of the Device:2 Data Model (if any) are not supported by the device type.

```
<dt:document xmlns:dt="urn:broadband-forum-org:cwmp:devicetype-1-1" ...
             deviceType="urn:your-company-com:some-device-type-id">
  <annotation>...</annotation>

  <import file="tr-181-2-2.xml" spec="urn:broadband-forum-org:tr-181-2-2">
    <model name="Device:2.2"/>
  </import>
  ...
  <model ref="Device:2.2">
    ...
  </model>
```

[27] This is necessary because the SupportedDataModel table (SDM) includes a parameter for the DT URL and a parameter for the DM spec. If the CPE combined its vendor-specific and standard model into one DT instance (i.e. only one entry in the SDM table), then there could only be one spec value specified in the SDM table (i.e. it would not be able to indicate both specs). For example, if SDM.1.URN is the vendor-specific spec value then there is no way to indicate what the standard model spec value is. The assumption is that a vendor-specific spec value would be different from a standard model spec value.

 © The Broadband Forum. All rights reserved

```
</dt:document>
```

The `import/@file` attribute indicates the name of the import file. The `import/@spec` attribute (optional) indicates the spec value of the import file. The `import/model/@name` attribute indicates the name and version of the model to be imported from the import file.

© The Broadband Forum. All rights reserved

7.3 Model

The top-level `model` element (II.6) is used to specify how the device type will support an associated DM Data Model. It will contain the subset of items (e.g. objects and parameters) that are supported, and details of how these items are supported differently (usually a "narrowing" in scope) from their DM definitions. Only one such top-level `model` element is allowed per DT Instance.

The model element will specify the name and version of its associated DM Data Model. This is done using the `model/@ref` attribute, where the name and version are represented as a single value in the form Name:Major.Minor (e.g. Device:2.2).

Note that a model element need not contain the entire supported Data Model for a given DM Data Model. This is because multiple DT Instances can each specify a portion of a supported Data Model that together combine to define the entire supported Data Model for a given DM Data Model. However, if an object or parameter is absent from all DT Instance models then it is not supported by the device type.

Note – The `model` element must reference the single DM model associated with the document; i.e. that was imported via an `import/model` element (see Section 7.2.2).

7.3.1 Supporting One DM Model

A DT model is based on a specific version of an associated DM model (e.g. Device:2.2, STBService:1.1, etc.). The DM model must first be imported into the DT Instance document, and then it can be referenced by the DT model via its `model/@ref` attribute.

Note that the referenced DM model should be the highest revision supported by the device type (i.e. earlier revisions of a DM model need not be referenced as they are automatically included as part a DM model's later revision).

Support for a given DM model can either be specified within one DT Instance document, or across multiple DT Instance documents all referencing the same DM model. The second option provides a modular approach in specifying support for a DM Data Model.

The following example illustrates support for the Device:2.2 Data Model (indicated by the `model/@ref` attribute) within a single DT Instance. Note that only one DM model is imported.

```
<dt:document xmlns:dt="urn:broadband-forum-org:cwmp:devicetype-1-1" ...
             deviceType="urn:your-company-com:some-device-type-id">
  <annotation>Supported Data Model for Device:2.2</annotation>

  <import file="tr-181-2-2.xml" spec="urn:broadband-forum-org:tr-181-2-2">
    <model name="Device:2.2"/>
  </import>
  ...

  <model ref="Device:2.2">
    ...
  </model>
</dt:document>
```

 © The Broadband Forum. All rights reserved

7.3.2 Supporting Multiple DM Models

A device type's support for different DM models must be specified within different DT Instance documents, following the same guidance outlined in Section 7.3.1 above. This requirement is borne out by TR-157 [5], in the DeviceInfo.SupportedDataModel.{i} table description.

The following example illustrates support for the Device:2.2 and STBService:1.1 Data Models. This requires at least two DT Instance documents since they can each only be associated with a single DM model.

```
<dt:document xmlns:dt="urn:broadband-forum-org:cwmp:devicetype-1-1" ...
             deviceType="urn:your-company-com:some-device-type-id">
  <annotation>Supported Data Model for Device:2.2</annotation>

  <import file="tr-181-2-2.xml" spec="urn:broadband-forum-org:tr-181-2-2">
    <model name="Device:2.2"/>
  </import>
  ...

  <model ref="Device:2.2">
    ...
  </model>
</dt:document>
```

```
<dt:document xmlns:dt="urn:broadband-forum-org:cwmp:devicetype-1-1" ...
             deviceType="urn:your-company-com:some-device-type-id">
  <annotation>Supported Data Model for STBService:1.1</annotation>

  <import file="tr-135-1-1.xml" spec="urn:broadband-forum-org:tr-135-1-1">
    <model name="STBService:1.1"/>
  </import>
  ...

  <model ref="STBService:1.1">
    ...
  </model>
</dt:document>
```

 © The Broadband Forum. All rights reserved

7.4 Object

The `object` element (II.7) is used to indicate support for an object that is defined in the associated DM Data Model. In addition, it can indicate how this object is supported differently (usually a “narrowing” in scope) from its DM definition.

All supported objects will be specified within some DT Instance. If an object defined in a DM Data Model is omitted from the DT model(s), then this means that the device type does not support this object.

As in the DM Data Model, objects are logically organized into an object hierarchy. The DT object hierarchy will follow the same layout as its associated DM Data Model (minus the unsupported objects).

7.4.1 Supporting Single-Instance Objects

Support for a single-instance object is specified within a model using the `object` element (II.7). Its `object/@ref` attribute specifies the path name of the corresponding DM object to be supported.

The object specification will include the following required attributes: `ref`, `access`, `minEntries`, and `maxEntries`. For single-instance objects the access (from the ACS point of view) will always be readOnly, minEntries can be 0 or 1, and maxEntries will always be 1. Basically, these attributes will have the same value in the DT object as they do in the corresponding DM object definition (i.e. none of these attributes can be supported differently).

The following example illustrates support for the Device object. Note that its path name corresponds exactly to an existing object in the DM Data Model.

```
<dt:document xmlns:dt="urn:broadband-forum-org:cwmp:devicetype-1-1" ...
             deviceType="urn:your-company-com:some-device-type-id">
  ...
  <model ref="Device:2.2">
    ...
    <object ref="Device." access="readOnly" minEntries="1" maxEntries="1">
      ...
    </object>
    ...
  </model>
</dt:document>
```

Note – An object will also contain parameter elements; this is not shown in the above example. See Section 7.5 for further details.

7.4.2 Supporting Multi-Instance Objects

Support for a multi-instance object (table) is specified within a model using the `object` element (II.7). Its `object/@ref` attribute specifies the path name of the corresponding DM object to be supported.

© The Broadband Forum. All rights reserved

The object specification will include the path name, plus an indication of whether the object is read-only or read-write (from the ACS point of view), and the number of instances of that object that can exist within the CPE. This corresponds to the following required attributes: `ref`, `access`, `minEntries`, and `maxEntries`.

The `object/@access` attribute indicates whether the table will have read-only or read-write access. Valid values include one of: readOnly, create, delete, createDelete. When `object/@access` is "readOnly" the ACS can only retrieve object instances from the CPE. When `object/@access` is "create" the ACS can retrieve and create, but not delete object instances. When `object/@access` is "delete" the ACS can retrieve and delete, but not create object instances. When `object/@access` is "createDelete" the ACS can retrieve, create, and delete object instances.

For a multi-instance object, the following restrictions apply:

- If the associated DM object has readWrite access, then the DT object can have any of the aforementioned access values. However, if the associated DM object is readOnly, then the DT object must also be readOnly. This means that read-only DM objects must remain read-only in the DT, but that writable DM objects can be restricted via the DT in how they are accessed by the ACS.
- The `object/@minEntries` and `object/@maxEntries` attributes must be specified such that their integer values fall within the min-maxEntries range specified by the DM object (i.e. restricting the range of min-maxEntries in the DT object, not expanding it). So the DT minEntries will be greater or equal to the DM minEntries, and the DT maxEntries will be less or equal to the DM maxEntries (with the proviso that minEntries must be less than maxEntries, and that all values are regarded as being less than an "unbounded" maxEntries).

The following example illustrates support for various tables in the InternetGatewayDevice:1.2 Data Model. Each object references a path name that corresponds to an existing object in the DM Data Model.

```
<dt:document xmlns:dt="urn:broadband-forum-org:cwmp:devicetype-1-1" ...
             deviceType="urn:your-company-com:some-device-type-id">
  ...
  <model ref="InternetGatewayDevice:1.2">
    ...
    <object ref="InternetGatewayDevice.DeviceInfo.VendorConfigFile.{i}."
            access="readOnly" minEntries="1" maxEntries="unbounded">
      ...
    </object>

    <object ref="InternetGatewayDevice.Layer3Forwarding.Forwarding.{i}."
            access="delete" minEntries="0" maxEntries="unbounded">
      ...
    </object>

    <object ref="InternetGatewayDevice.LANDevice.{i}.WLANConfiguration.{i}.WEPKey.{i}."
            access="readOnly" minEntries="4" maxEntries="4">
      ...
    </object
    ...
  </model>
</dt:document>
```

 © The Broadband Forum. All rights reserved

In the above example, we see support specified for the following tables:

- VendorConfigFile.{i} – In the DM this is defined as a read-only, variable-sized table that can have 0 or more entries (min entries is 0, max entries is unbounded). However, in the DT this object is supported with min entries of 1; i.e. a CPE of this device type will always have at least one vendor config file.
- Forwarding.{i} – In the DM this is defined as a writable (access is readWrite), variable-sized table that can have 0 or more entries. However, in the DT this object's access is delete; i.e. a CPE of this device type allows the ACS to retrieve and delete, but not create these object instances.
- WEPKey.{i} – In the DM this is a read-only, fixed-sized table (min entries equals max entries, which is 4). In the DT this object is specified the same as its corresponding DM object definition, meaning that it is supported as is without change.

Note – An object will also contain parameter elements; this is not shown in the above example. See Section 7.5 for further details.

 © The Broadband Forum. All rights reserved

7.5 Parameter

The `parameter` element (II.8) is used to indicate support for a parameter that is defined in the associated DM Data Model. In addition, it can indicate how this parameter is supported differently (usually a "narrowing" in scope) from its DM definition.

All supported parameters will be specified within some DT Instance. If a parameter defined in a DM Data Model is omitted from the DT model(s), then this means that the device type does not support this parameter.

7.5.1 Supporting Parameters (The Basics)

Support for a parameter is specified within a model or object using the `parameter` element (II.8). Its `parameter/@ref` attribute specifies the name of the corresponding DM parameter to be supported.

The parameter specification will include the name, plus an indication of whether the parameter is read-only or read-write, its active notify setting (optional), and the syntax for valid parameter values (optional). This corresponds to the following attributes and elements: `ref`, `access`, `activeNotify`, `syntax`. Note that `ref` and `access` are required attributes, while the others are optional.

Note – Omission of an optional element or optional attribute results in the DT parameter supporting the maximum that the DM definition permits for that element or attribute. For example, if the DT parameter does not specify a `syntax` element, then the full range of syntax permitted by the DM definition is supported.

The following restrictions also apply:

- The `parameter/@access` attribute indicates whether the parameter will have read-only or read-write access (valid values are: readOnly, readWrite). If the associated DM parameter has readWrite access, then the DT parameter can be either readOnly or readWrite. However, if the associated DM parameter is readOnly, then the DT parameter must also be readOnly. This means that a CPE can choose to offer read-only access to a parameter for which the DM definition indicates write access makes sense.
- The `parameter/@activeNotify` attribute indicates the parameter's notification policy (this is an optional attribute; valid values are: normal, willDeny). Normal means the DT parameter supports the value defined in the DM parameter. If the DM parameter is set to canDeny then the DT parameter can be further restricted by setting its activeNotify to willDeny.
- The supported syntax can be specified either using one of the built-in primitive data types or using a named data type. This is discussed in Section 7.5.1.1 below. The `syntax` element is optional since it is common for a CPE to omit it in order to support the maximum syntax that the DM definition permits (i.e. the full range, maximum size, all the enumeration values, etc.).

The following example illustrates support for the EnableCWMP parameter. In the DM this is defined as a readWrite parameter. However, in the DT example this parameter is supported with

 © The Broadband Forum. All rights reserved

readOnly access; i.e. a CPE of this data type will not allow an ACS to change the parameter's value. Note that the parameter's syntax element has been omitted.

```
<dt:document xmlns:dt="urn:broadband-forum-org:cwmp:devicetype-1-1" ...
             deviceType="urn:your-company-com:some-device-type-id">
  ...
  <model ref="Device:2.2">
    ...
    <object ref="Device.ManagementServer." ...>
      ...
      <parameter ref="EnableCWMP" access="readOnly"/>
      ...
    </object>
    ...
  </model>
</dt:document>
```

The following example illustrates support for the ParameterKey parameter. In the DM this is defined as a readOnly parameter with activeNotify set to canDeny. However, in the DT example this parameter is supported with an activeNotify of willDeny. Note that the DT cannot use a different value for the `access` attribute since it is readOnly in the DM.

```
<dt:document xmlns:dt="urn:broadband-forum-org:cwmp:devicetype-1-1" ...
             deviceType="urn:your-company-com:some-device-type-id">
  ...
  <model ref="Device:2.2">
    ...
    <object ref="Device.ManagementServer." ...>
      ...
      <parameter ref="ParameterKey" access="readOnly" activeNotify="willDeny"/>
      ...
    </object>
    ...
  </model>
</dt:document>
```

7.5.1.1 Supported Syntax and its Data Type

A parameter value's supported syntax is defined using the parameter `syntax` element (II.8.1). This element can specify a parameter's supported data type and range of permitted values.

Note – A DT parameter's `syntax` element (and its contained data type) is optional. They can be omitted as discussed in Section 7.5.1.

If a device type wishes to restrict or extend a DM parameter's data type definition, then there are two ways to specify the DT parameter's supported data type: either via a built-in primitive data type or via a reference to a named data type. They are mutually exclusive.

Using either a primitive data type or a named data type:

- Primitive data types – The DM Schema defines a set of built-in primitive data type elements that can be used in specifying a DT parameter's supported syntax; i.e. one of: base64, boolean, dateTime, hexBinary, int, long, string, unsignedInt, unsignedLong (see Section II.9 for details on each). One of these elements can appear within a DT parameter's syntax element (e.g. parameter/syntax/boolean); a parameter's syntax element can only contain one such primitive data type element.

 © The Broadband Forum. All rights reserved

- Named data types – A DT parameter's syntax can reference a named data type using the `dataType` element (II.8.3). The reference is made via its `ref`[28] or `base` attribute (e.g. `parameter/syntax/dataType/@base="IPAddress"`). The referenced data type must already be defined elsewhere in some DM Instance; see Section 6.2 for details on defining named data types.

Note that the DT parameter's base data type (e.g. boolean, IPAddress, etc.) must be the same as defined in the corresponding DM parameter. The point is to alter the supported data type in some basic way, not to redefine it entirely.

Therefore, specifying a data type within the DT parameter syntax is only the first step. Now we need to specify how the data type will be supported differently (restricted or extended) from the DM definition. This difference is specified using facets (see Section 7.5.1.2 below for further details).

7.5.1.2 Supported Data Type Using Facets

Note – This section builds on the discussion in Section 7.5.1.1 above. An understanding of supported syntax and data types are required in order to use facets.

A parameter's supported data type can be refined using facets. Facet elements are used, within a parameter's primitive or named data type element, to specify support for a data type that is a restriction or extension of its corresponding DM definition (e.g. a smaller string size or numeric range). When specified within a parameter that references a named data type, the `dataType/@base` attribute must be used.

DT facet use is analogous to facets used within a DM parameter's data type definition (6.6.1.3). However, the full set of DT facet elements is: `size`, `range`, `pattern`, `enumeration`, `pathRef`. See Section II.10 for details on using each of these facets.

Note – If a facet is omitted from a DT parameter, but is present in the corresponding DM parameter definition, the implication is that the DT parameter will support the maximum that the DM facet definition permits. However, when such a facet is included, it must be fully specified in order to completely define the facet within the DT parameter.

Note that not all facets can be used within all primitive data types. This is also the case within named data types, since they are derived from primitive data types. Section II.9 Table 74 lists which facets are valid with each primitive data type.

The following example illustrates support for a URL parameter and a CWMPRetryMinimum-WaitInterval parameter (using facets within primitive data type elements). In the DM, the URL parameter is defined as a string with maximum length 256 (using the `size` facet), and the other parameter is defined as an unsigned integer with range 1 to 65535 (using the `range` facet).

[28] The `dataType/@ref` attribute will be defined in DT Schema v1.2. This will permit a DT parameter's data type to be referenced while still allowing it to be supported as defined in the DM parameter definition.

 © The Broadband Forum. All rights reserved

However, in this DT example the URL parameter's max length is restricted to 128, and the other parameter's max range is restricted to 255.

```
<dt:document xmlns:dt="urn:broadband-forum-org:cwmp:devicetype-1-1" ...
             deviceType="urn:your-company-com:some-device-type-id">
  ...
  <model ref="Device:2.2">
    ...
    <object ref="Device.ManagementServer." ...>
      ...
      <parameter ref="URL" access="readWrite">
        <syntax>
          <string>
            <size maxLength="128"/>
          </string>
        </syntax>
      </parameter>

      <parameter ref="CWMPRetryMinimumWaitInterval" access="readWrite">
        <syntax>
          <unsignedInt>
            <range minInclusive="1" maxInclusive="255"/>
          </unsignedInt>
        </syntax>
      </parameter>
      ...
    </object>
    ...
  </model>
</dt:document>
```

The next example snippet demonstrates the use of a facet within a parameter that has a named data type (i.e. within the `parameter/syntax/dataType` element). This example assumes that String255 is a named data type that has been defined as a string with max length 255. Use of the `dataType/@base` attribute is required to reference the named data type. The `size` facet is then used to restrict the parameter's max string length down to 127 characters.

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

7.5.1.3 Default Value

A default parameter value is specified using the parameter syntax's `default` element (II.8.4). This element can indicate either a factory default or an object default (one or the other).

Note that a DT parameter's default value can only be specified if one was not already defined by the corresponding DM parameter. In other words, a supported default is specified in order to define a missing default, not to change an existing default in the DM definition.

An object default should only be used with parameters that come about due to object creation. A factory default can potentially be used with any parameter, but only applies when the parameter's default value is based on some standard, e.g. RFC.

 © The Broadband Forum. All rights reserved

The type of default is indicated by the required attribute `parameter/syntax/-default/@type`. Possible values for this attribute are: factory or object. The default value itself is indicated by the required attribute `parameter/syntax/default/@value`. This value must be valid for the parameter's data type.

The following example illustrates the STUNEnable parameter with a factory default. Note that this parameter has no default value in the DM object definition, and that the default value (true) is valid for the parameter's boolean data type.

```
<dt:document xmlns:dt="urn:broadband-forum-org:cwmp:devicetype-1-1" ...
             deviceType="urn:your-company-com:some-device-type-id">
  ...
  <model ref="Device:2.2">
    ...
    <object ref="Device.ManagementServer." ...>
      ...
      <parameter ref="STUNEnable" access="readWrite">
        <syntax>
          <boolean/>
          <default type="factory" value="true"/>
        </syntax>
      </parameter>
      ...
    </object>
    ...
  </model>
</dt:document>
```

Note that the default element if present must be the last element within the syntax.

7.5.2 List-Valued Parameter

Support for a list-valued parameter's list is altered using the parameter syntax `list` element (II.8.1/Table 67 and II.8.2). As discussed earlier, such a parameter is always a string, while its data type specification (int, boolean, etc.) applies to individual list items rather than the parameter value as a whole.

Note – It is not valid for a DT parameter to specify a `list` element unless its corresponding DM parameter is itself list-valued. When present, the parameter/syntax/list element will be the first element within the syntax.

The number of items supported in a `parameter/syntax/list` element is specified using its `minItems` and `maxItems` attributes. Both are optional attributes; neither, either, or both can be specified..

The `parameter/syntax/list/size` element can be used to explicitly specify the minimum and maximum length of the parameter's overall string value, via its `minLength` and `maxLength` attributes (see II.10.1). Alternatively, the maximum length can instead be implied by the maximum number of items and the individual item lengths (as discussed in Section 3.2.6/TR-106 [3]). The latter is preferred when the number and length of items is known.

© The Broadband Forum. All rights reserved

The following example illustrates support for a list-valued parameter whose overall value has a restricted max length. In the DM the parameter is defined with max length 256. However, in the DT the parameter's max length has been restricted to 128. Note that the `string` element could have been omitted since it is not restricting the DM definition via facets (it was just included for readability).

```
<parameter ref="AvailableLanguages" ...>
  <syntax>
    <list>
      <size maxLength="128"/>
    </list>
    <string/>
  </syntax>
</parameter>
```

The following example illustrates support for a parameter whose value is a list of MAC addresses. In the DM definition the list's max number of items is 16. However, in the DT the list's max number of items is restricted to 8. Again, the `dataType` element could have been omitted since it is not restricting the DM parameter definition via facets.

```
<parameter ref="EndStationMACs" ...>
  <syntax>
    <list maxItems="8"/>
    <dataType ref="MACAddress">
  </syntax>
</parameter>
```

In the above example the list's max length is not specified, but it is implied given its max items and the fact that a MACAddress has a max length.

7.5.3 Path-Reference Parameter

Support for a path-reference parameter's reference is altered using the `pathRef` facet (II.10.2). As discussed in the corresponding DM section (6.6.6.1), a path-reference parameter references another parameter or object via its path name.

Note – It is not valid for a DT parameter to specify a `pathRef` facet unless its corresponding DM parameter is itself defined as a path-reference.

The `pathRef` facet element has several attributes that aid in specifying which parameters and objects will be targeted as candidates for the reference. These are: `targetParent`, `targetType`, and `targetDataType`. Note that these attributes are used to narrow what is already defined in the DM definition; it is not valid for a DT parameter specification to broaden which parameters and objects can be targeted.

The set of parameters and objects that can be referenced is restricted using the optional `pathRef/@targetParent` attribute. This attribute contains a list of object path names; only the immediate children of one of these specified (parent) objects can be referenced. This will be a subset of the items specified in the DM targetParent attribute.

 © The Broadband Forum. All rights reserved

The type of item that can be referenced is restricted using the optional pathRef/-@targetType attribute. This can be one of: any (i.e. parameter or object), parameter, object, single (i.e. single-instance object), table (i.e. multi-instance object), row (i.e. of a table).

When the item to be referenced is a parameter, then the type of parameter referenced can be restricted using the optional pathRef/@targetDataType attribute. This is relevant only when targetType is configured for parameters (i.e. any or parameter). Possible values for targetDataType are: any (any data type), base64, boolean, dateTime, hexBinary, integer (any numeric data type), int, long, string, unsignedInt, unsignedLong, or the name of some named data type.

The following example illustrates support for a pathRef parameter whose targetParent has been restricted. In the DM the parameter is defined with a targetParent that includes both the Bridge and VLAN tables. However, in the DT the parameter's targetParent has been restricted to just the Bridge table. The other pathRef attributes, targetType and targetDataType, are specified just as they were specified in the DM (i.e. targetType is row, and targetDataType is omitted); this is to ensure that the pathRef definition is fully specified.

```
<object ref="Device.Bridging.Filter.{i}." ...>
  ...
  <parameter ref="Bridge" ...>
    <syntax>
      <string>
        <pathRef targetParent="#.Bridge." targetType="row"/>
      </string>
    </syntax>
  </parameter>
  ...
</object>
```

The following is another example that illustrates support for a pathRef parameter whose targetParent has been restricted. In the DM the parameter is defined with no explicit targetParent (i.e. defaults to an empty list indicating any target parent). However, in the DT the parameter's targetParent has been restricted to only those object references that the device type supports.

```
<object ref="Device.InterfaceStack.{i}." ...>
  ...
  <parameter ref="HigherLayer" ...>
    <syntax>
      <string>
        <pathRef targetParent=".Bridging.Bridge.{i}.Port. .Ethernet.Link.
                              .PPP.Interface. .IP.Interface."
                 targetType="row"/>
      </string>
    </syntax>
  </parameter>
  ...
</object>
```

In the above example, the use case illustrated is the DM says "it references something but it cannot list all the possibilities, e.g. because it could be any interface object", whereas DT says "it references something and these are the things that it can reference".

 © The Broadband Forum. All rights reserved

7.6 Annotation

An annotation is free text used to describe implementation specific details (similar to DM descriptions). It can contain a limited amount of MediaWiki-like markup that processing tools can interpret. Annotations will use the same markup as specified for DM descriptions (see Appendix I.2.2 and I.2.3).

Many different types of DT elements (e.g. object, parameter) can contain an annotation. An annotation is specified using an `annotation` element (II.2) within the element to be described (e.g. an `object/annotation` element describes its object parent). If the parent element has other sub-elements, the annotation must be the first element to appear within the parent element.

To determine whether a particular element can have an annotation, see its corresponding reference section in Appendix II.

The following example illustrates a model annotation and an object annotation. The annotations are simply free text within an annotation element. As required, each annotation is the first element within its parent element.

```
<dt:document xmlns:dt="urn:broadband-forum-org:cwmp:devicetype-1-1" ...
             deviceType="urn:your-company-com:some-device-type-id">
  ...
  <model ref="Device:2.2">
    <annotation>Some implementation specific comment regarding the model<annotation/>
    ...
    <object ref="Device.IP.Interface.{i}." ...>
      <annotation>Some comment about the device type’s support of this object with markup and
                  a bibref Template {{bibref|TR-181i2|Section 4.2}}).</annotation>
      ...
    </object>
    ...
  </model>
</dt:document>
```

Note – The above annotation example includes a `{{bibref}}` Template (markup) used to reference a document from the imported bibliography. See Section 6.1.3 for a discussion on citing bibliographic references.

© The Broadband Forum. All rights reserved

7.7 Feature

Support for a feature is specified using the top-level `feature` element (II.3). Its `feature/@name` attribute specifies the name of a feature supported from the DM model. Use of features is a shorthand way of broadly indicating what a DT model supports.

A CPE's feature list is the union of all the features in all of its DT Instances.

The following well-known feature names are defined:

- DNSClient
- DNSServer
- Firewall
- IPv6
- NAT
- Router

These names are not defined in the DM; rather they are a static list of general features defined by the separate DTF Schema (Appendix III). Vendor-specific feature names are also permitted within the DT `feature/@name` attribute (i.e. names prefixed with X_<VENDOR>_, as specified by Section 3.3/TR-106 [3]).

Note – Vendor-specific feature names are defined simply by using them in a `feature` element. Since they are not a DM-level concept, they do not have to be defined in a vendor-specific DM Instance before being referenced in a DT Instance.

The following example illustrates the use of features. As required, features appear immediately before the model element. The vendor-specific feature name X_ACDC73_VPN is defined simply by listing it.

```
<dt:document xmlns:dt="urn:broadband-forum-org:cwmp:devicetype-1-1" ...
             deviceType="urn:your-company-com:some-device-type-id">
  ...
  <feature name="IPv6"/>
  <feature name="Firewall"/>
  <feature name="X_ACDC73_VPN"/>

  <model ref="Device:2.2">
    ...
  </model>
</dt:document>
```

Note that specified features are not referenced throughout the document. They are just used as an indicator at the top of the document only.

© The Broadband Forum. All rights reserved

Appendix I – Reference: Data Model XML Schema

This appendix provides a user reference for the TR-069 Data Model schema (DM Schema), v1.4. The normative version can be found at http://www.broadband-forum.org/cwmp/cwmp-datamodel-1-4.xsd.

I.1 `document` Element

The `document` element is the root element of any DM XML Data Model file. It is required.

Table 1 lists the attributes that are available within the `document` element.

Table 1 – DM `document` attributes

Attribute	Type	Use	Description
spec	xs:anyURI	required	URI of the associated specification document, e.g. the BBF Technical Report. An empty string is not allowed. This URI should uniquely identify the specification (the document). More than one DM Schema instance document may reference the same specification. Where the specification is a BBF document, the URI naming rules specified in Section A.2.1.1/TR-106 must be used. For example, to reference TR-106 Issue 1 Amendment 2 Corrigendum 0, the value of this attribute would be urn:broadband-forum-org:tr-106-1-2-0 **Note** – The spec value will always include the corrigendum number. This is because it needs to uniquely identify a specification, and so omitting the corrigendum (which assumes the use of the latest corrigendum released) is not appropriate here.
file	xs:anyURI	required	File name (omitting directory name) of this document. An empty string is not allowed. Where the specification is a BBF document, the file naming rules specified in A.2.1.1 must be used. For example, for the Data Model defined in TR-106 Issue 1 Amendment 2 Corrigendum 0, the value of this attribute would be tr-106-1-2-0.xml. **Note** – The file value will always include the corrigendum number. This is because it needs to uniquely identify a file, and so omitting the corrigendum (which assumes the use of the latest corrigendum released) is not appropriate here.

 © The Broadband Forum. All rights reserved

The following example illustrates the use of the `document` element. It indicates the `file` and `spec` that the `document` purports to define and also specifies a `dm` namespace to be used throughout the document.

```
<dm:document xmlns:dm="urn:broadband-forum-org:cwmp:datamodel-1-4"
    spec="urn:broadband-forum-org:tr-181-2-3-0"
    file="tr-181-2-3-0.xml">
  …
</dm:document>
```

Note – In the above example, although outside the scope of this appendix, it is expected that the `document` element will also include the `xmlns:xsi` and `xsi:schemaLocation` attributes in order to specify the location of the XSD schema file to be associated with the `dm` namespace.

Table 2 lists the child elements allowed within a `document`. The order that these elements appear in the table is the same order, if present, that they must appear within a `document` definition (with the exception that `component` elements can appear before or after `model` elements).

Table 2 – DM `document` sub-elements

Element	Multiplicity	Description
description	0 or 1	Top-level description (I.2).
import	0 or more	Imported data types, components and model (Root and Service Objects) definitions (I.3).
dataType	0 or more	Top-level data type definitions (I.4).
bibliography	0 or 1	Bibliographic references (I.5).
component	0 or more	Component definitions (I.6).
model	0 or more	Model (Root and Service Object) definitions (I.8).

I.2 `description` Element

The `description` element holds free text (i.e. of type xs:string) which can contain a limited amount of MediaWiki-like markup as specified in Sections I.2.2 and I.2.3. For example, use "*" at the start of a line to indicate a bulleted list. For Broadband Forum standards, the character set will 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.

The `description` element can be used within almost every other element (e.g. under `document`, `document/model`, `document/model/object`, etc.). When used, it will always be the first element to appear under its parent element. Refer to specific sections within this appendix to determine whether or not the description element is permitted.

Table 3 lists the attributes that are available within the `description` element.

© The Broadband Forum. All rights reserved

Table 3 – DM `description` attributes

Attribute	Type	Use	Description
action	One of: • create • prefix • append • replace	optional	The default is *create* if not specified. This must be specified when the description modifies that of a previously defined item. While *create* will only ever be specified when the description is for an initially defined item. Specify “prefix” to prefix to the parent element’s previous description, "append" to append to the previous description, or "replace" to replace the previous description.

The following example illustrates the syntax used to replace the description previously defined for some parent element.

```
<description action="replace">
  This object defines the…
</description>
```

I.2.1 Whitespace Pre-processing

Note – These pre-processing rules are largely copied from Section A.2.2.2/TR-106 [3].

Processing tools, such as the Report Tool, will discard leading and trailing whitespace from descriptions prior to interpreting any markup within said description. The following rules apply:

- Discard any leading whitespace up to and including the first line break.
- Discard the longest common whitespace prefix (i.e. that occurs at the start of every line) from each line. In the example below, three lines start with four spaces and one line starts with five spaces, so the longest prefix to be removed from the start of each line is four spaces. In this calculation, a tab character counts as a single character, so to avoid confusion, the description should not contain tab characters.
- Discard 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.
```


© The Broadband Forum. All rights reserved

```
    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.
```

I.2.2 Markup

Note – Table 4 is largely copied from Section A.2.2.3/TR-106 [3].

A description can contain the following markup, which is inspired by, but is not identical to, MediaWiki markup. Processing tools are expected to interpret this markup as indicated below (e.g. the Report Tool interprets the markup when generating its HTML reports).

Table 4 – 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.
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.

 © The Broadband Forum. All rights reserved

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 Section A.2.2.2/TR-106 [3] 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.
		Section I.2.3 specifies the standard Templates.

The following example illustrates a description element that contains markup. It has a `{{bibref}}` Template that references the `bibliography` element named ‘example’, the word ‘one’ is italicized, and the items {red, green, blue} are in a bullet list.

```
<description>
  {{bibref|example}} requires the use of ''one'' of the following items:
  * red
  * green
  * blue
</description>
```

Note – See Section A.2.3.5/TR-106 [3] for a more complete example, which includes a description element showing most markup as well as the resulting HTML that the Report Tool is expected to generate.

I.2.3 Templates

Note – Table 5 is largely copied from Section A.2.2.4/TR-106 [3].

A Template is a kind of markup. It is encoded within a description 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}}`

Processing tools are expected to replace Template references with template-dependent text (e.g. the Report Tool replaces each Template with appropriate text when generating its HTML reports).

The following standard Templates are defined. Any vendor-specific template names must obey the rules of Section 3.3/TR-106.

© The Broadband Forum. All rights reserved

Table 5 – 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 (see Section A.2.4/TR-106 [3]).
		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}}`

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 Section A.2.3.4/TR-106 [3].
		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

© The Broadband Forum. All rights reserved

		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 (see Section A.2.7.1/TR-106 [3]).
		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>.".

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 (see Section A.2.3.7/TR-106 [3]).
		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

 © The Broadband Forum. All rights reserved

		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 (see Section A.2.3.1/TR-106 [3]).
		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 (see Section A.2.9/TR-106 [3]).
		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).

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 Section A.2.3.4/TR-106 [3]. 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

 © The Broadband Forum. All rights reserved

		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 Section A.2.3.4/TR-106 [3]. 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}}`

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 (see Section A.2.7.1/TR-106 [3]).
		This is a hint to processing tools to replace the Template reference with text explaining that the value

© The Broadband Forum. All rights reserved

		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 (see Section A.2.7.1/TR-106 [3]).
		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 (table) object that defines one or more unique keys (see Section A.2.8.1/TR-106 [3]).
		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.

© The Broadband Forum. All rights reserved

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.
Misc.	{{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 (see Section A.2.3.5/TR-106 [3]), e.g. {{empty}}, {{false}} or 0.
	{{issue|descr}} {{issue|label|descr}} {{issue|label|RESOLVED| descr}}	Signifies an open issue. This is a hint to processing tools to replace the Template reference with a clearly visible rendition of the open issue. The descr argument describes the issue. The optional label can be used to label the issue (e.g. different labels could be used for different categories of issue) and the optional status can be set to RESOLVED to indicate that the issue has been resolved. This Template can be used within any description.

Note – The {{issue}} Template is not mentioned in Section A.2.2.4/TR-106 [3]. This is because it is only intended to be utilized while a document is a draft. All {{issue}} Templates need to be removed from a document prior to its publication.

I.3 import Element

The import element is used to import data types, components and models (Root and Service Objects) from external documents.

By importing such item definitions, they are then available to be referenced throughout the local document rather than having to redefine them. However, if these imported items are not subsequently referenced, then they do not actually contribute to the local document's Data Model (i.e. importing items from an external file does not automatically make them part of the model defined within the local document).

The import mechanism is recursive; i.e. if an imported file itself includes imports, then these imports are also available in the local document (as is the case with the imported file's imported files, and so on).

Possible parent elements include:

- document (I.1)

 © The Broadband Forum. All rights reserved

Table 6 lists the attributes that are available within the `import` element.

Table 6 – DM `import` attributes

Attribute	Type	Use	Description
file	xs:anyURI	required	URI of the file. It must be used to locate the DM Instance (see Section A.2.1/TR-106). **Note** – The file value will omit the corrigendum.
spec	xs:anyURI	optional	URI of the spec. If specified, processing tools must regard a mismatch between this and the external document's spec attribute as an error. **Note** – The spec value will omit the corrigendum.

The following example illustrates how to reference an external document. Specific items to be imported from this file are specified using `import` sub-elements.

```
<import file="tr-143-1-0.xml" spec="urn:broadband-forum-org:tr-143-1-0">
  …
</import>
```

Table 7 lists the child elements allowed within an `import`. These are explained in greater detail in the next section.

Table 7 – DM `import` sub-elements

Element	Multiplicity	Description
dataType	0 or more	Reference to a `dataType` in the external document (I.3.1).
component	0 or more	Reference to a `component` in the external document (I.3.1).
model	0 or more	Reference to a `model` in the external document (I.3.1).

I.3.1 `import` Sub-elements

The `import` element has three possible child elements: `dataType`, `component`, and `model`. Each of these is used to indicate specific items to be imported from the external document that is specified within the parent `import` element.

Possible parent elements include:

- `document/import` (I.3)

Table 8 lists the attributes that are available within the `dataType`, `component`, and `model` sub-elements.

Table 8 – DM `import`'s sub-element attributes

Attribute	Type	Use	Description
name	restricted xs:NCName	required	Name of the imported item as it is known within the local document.
ref	restricted	optional	Name of the item within the external document. This

 © The Broadband Forum. All rights reserved

	xs:NCName		attribute is used when the imported item is known by different names within the local and external documents. If the `ref` attribute is omitted, then the item being imported is also known by this name within the external document.

The following example imports the IPAddress data type from an external document. Since the `dataType/@ref` attribute is omitted, this data type is expected to be known by IPAddress within the local and external documents.

```
<import file="tr-106-1-0-types.xml" spec="urn:broadband-forum-org:tr-106-1-0">
  <dataType name="IPAddress"/>
  …
</import>
```

Note – The format of the `dataType/@name` and `dataType/@ref` attributes is the same as xs:NCName, except that it cannot start with a lower-case letter (to avoid conflict with built-in data type names) and dots are not permitted.

The following example imports the DownloadDiagnostics component from an external document. However, within the local document the component will be known as _DownloadDiagnostics. Renaming an imported component in this way is necessary when the referenced name is already present within the local document (i.e. to avoid a name conflict).

```
<import file="tr-143-1-0.xml" spec="urn:broadband-forum-org:tr-143-1-0">
  <component name="_DownloadDiagnostics" ref="DownloadDiagnostics"/>
  …
</import>
```

Note – The format of the `component/@name` and `component/@ref` attributes is the same as xs:NCName, except that dots are not permitted.

The following example imports the Device:1.3 model from an external document. Since the `model/@ref` attribute is omitted, this model is expected to be known by Device:1.3 within the local and external documents.

```
<import file="tr-157-1-0.xml" spec="urn:broadband-forum-org:tr-157-1-0">
  <model name="Device:1.3"/>
  …
</import>
```

Note – The `model/@name` and `model/@ref` attributes include a name part, a colon, major version digits, a dot, and minor version digits. The format of the name part is the same as xs:NCName, except that dots are not permitted.

Of course, although not shown here, an `import` example could be crafted that imports data types, components, and models all from the same external document.

© The Broadband Forum. All rights reserved

I.4 `dataType` Element (definition)

The top-level dataType element is used to define named data types that can be referenced from parameter definitions or from other data type definitions. These named data types are defined using built-in primitive data types, other named data types, and data type facets, providing a means to define custom types beyond just the built-in primitive types.

Possible parent elements include:

- document (I.1)

Zero or more dataType elements are permitted within a document.

Table 9 lists the attributes that are available within the dataType element.

Table 9 – DM dataType attributes

Attribute	Type	Use	Description
name	restricted xs:NCName	required	The data type name. It must be unique within the document, including imported data types.
base	restricted xs:NCName	optional	The base type name; i.e. the name of an existing top-level data type definition from which this data type is derived. The base type name cannot be one of the built-in primitive types, but rather, must be the name of a dataType definition.
status	One of: • current • deprecated • obsoleted • deleted	optional	The status of the data type. The values *deprecated* and *obsoleted* are as described in Section 2.2/TR-106 [3]; *deleted* indicates the element is no longer present in the current Data Model; *current* indicates the element is present in the current Data Model. The default is *current* if not specified.
id	xs:token (256)	optional	Corresponds with the identifier from SMNP MIBs, included to allow easier import of these MIBs. Rarely used.

Note – The name and base attributes have the same format as xs:NCName except that they cannot begin with a lower-case letter (to avoid confusion with built-in primitive data type names) and dots are not permitted; i.e. must start with an upper-case letter or "_", and subsequent characters can also include digits and connector characters such as underscore and dash.

Table 10 lists the elements allowed within a dataType. The description element if present must appear first.

© The Broadband Forum. All rights reserved

Table 10 – DM dataType sub-elements

Element	Multiplicity	Description
description	0 or 1	The data type's description (I.2).
size instanceRef pathRef range enumeration enumerationRef pattern units	0 or more	Data type facets (I.13). These elements specify some aspect of a data type, e.g. its size, range, units, etc. Use of these elements is only permitted when the base type is a named data type (i.e. when the base attribute is specified). In this case, the primitive elements below are not used.
base64 boolean dateTime hexBinary int long string unsignedInt unsignedLong	0 or 1	Built-in primitive data types (I.12). One of these elements must be used when the base type is primitive (i.e. when the base attribute is omitted). When the base type is not primitive, then these elements are not permitted. The facet elements listed above are not used with a primitive type; instead, each primitive type has its own set of supported sub-element facets.

The following example defines the data type named String255. Since the base attribute is omitted, this definition needs to be based on a primitive type and therefore includes a primitive type element, in this case the string sub-element. Each primitive element supports its own set of child facets; here, the string maxLength is set to 255.

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

The following example defines the data type named String127. Its base attribute indicates that the base type is String255 (i.e. String127 is derived from String255). The size facet restricts the maximum length to 127, where it had been 255 in the base type. Since this is not a primitive type, the primitive child elements are not permitted.

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

I.5 bibliography Element

The bibliography element is used to define bibliographic references for various publications that might be cited throughout the document. This is also discussed in Section A.2.4/TR-106 [3].

© The Broadband Forum. All rights reserved

Possible parent elements include:

- `document` (I.1)

Table 11 lists the elements allowed within a `bibliography`. The order that these elements appear in the table is the same order, if present, that they must appear within a `bibliography` definition.

Table 11 – DM `bibliography` sub-elements

Element	Multiplicity	Description
description	0 or 1	The bibliography's description (I.2).
reference	0 or more	Information about the referenced publication; i.e. name, title, date, hyperlink, etc. (I.5.1).

I.5.1 Bibliography `reference` Element

The `bibliography` element's `reference` sub-element is used to specify the details regarding a particular publication. A publication can be referenced from descriptions throughout the document using the `{{bibref}}` Template (see I.2.3).

Possible parent elements include:

- `document/bibliography` (I.5)

Table 12 lists the attributes that are available within the `reference` element.

Table 12 – DM bibliography `reference` attributes

Attribute	Type	Use	Description
id	xs:token	required	Uniquely identifies the bibliographic reference within the document (and should uniquely identify this reference across all instance documents). For BBF DM Instances, the bibliographic reference ID rules specified in Section A.2.4/TR-106 must be used. For example, to reference TR-106 Issue 1 Amendment 2, the value of this attribute would be TR-106a2.

Table 13 lists the elements allowed within a `reference`. The order that these elements appear in the table is the same order, if present, that they must appear within a `reference` definition.

Table 13 – DM bibliography `reference` sub-elements

Element	Multiplicity	Type	Description
name	1	xs:token	Name by which the referenced document is usually known, e.g. TR-069, RFC 2863. This is a required element.
title	0 or 1	xs:token	Title of the referenced document.

© The Broadband Forum. All rights reserved

organization	0 or 1	xs:token	Organization that published the referenced document, e.g. BBF, IEEE, IETF.
category	0 or 1	xs:token	Document category, e.g. TR (BBF), RFC (IETF).
date	0 or 1	xs:token	Publication date.
hyperlink	0 or more	xs:anyURI	URI where the referenced document can be found.

The following example illustrates a bibliographic reference for RFC 2863.

```
<bibliography>
  <reference id="RFC2863">
    <name>RFC 2863</name>
    <title>The Interfaces Group MIB</title>
    <organization>IETF</organization>
    <category>RFC</category>
    <date>June 2000</date>
    <hyperlink>http://tools.ietf.org/html/rfc2863</hyperlink>
  </reference>
  …
</bibliography>
```

I.6 `component` Element (definition)

The top-level `component` element is used to define a group of objects, parameters and/or profiles that can later be included within some other definition. This should not be confused with a `component` element that appears inside components and models, which is instead a component reference used to include a component definition at that point within the document.

Possible parent elements include:

- `document` (I.1)

Table 14 lists the attributes that are available within this `component` element.

Table 14 – DM definition-based `component` attributes

Attribute	Type	Use	Description
name	restricted xs:NCName	required	Component name is unique within the document, including imported components **Note** – A `name` has the same format as xs:NCName except that dots are not permitted; i.e. must start with a letter or "_" and subsequent characters can also include digits and separator characters such as underscore or dash.
status	One of: • current • deprecated • obsoleted • deleted	optional	The status of the component. The values *deprecated* and *obsoleted* are as described in Section 2.2/TR-106 [3]; *deleted* indicates the element is no longer present in the current Data

© The Broadband Forum. All rights reserved

			Model; *current* indicates the element is present in the current Data Model. The default is *current* if not specified.
id	xs:token(256)	optional	Corresponds with the identifier from SMNP MIBs, included to allow easier import of these MIBs. Rarely used.

Table 15 lists the child elements allowed within this `component` element. The `description` element if present must appear first, while the `profile` elements if present must appear last. The `component`, `parameter`, and `object` elements can appear in any order just so long as they come after any `description` and before any `profile` elements.

Table 15 – DM definition-based `component` sub-elements

Element	Multiplicity	Description
description	0 or 1	The component's description (I.2).
component	0 or more	A component reference (I.7).
parameter	0 or more	A top-level parameter definition (I.10).
object	0 or more	An object definition (I.9).
profile	0 or more	A profile definition (I.11).

The following example defines a `component` named DeviceInfo. It illustrates some of the sub-elements that can be defined within a top-level component, such as object and parameter definitions.

```
<component name="DeviceInfo">
  <object name="DeviceInfo." access="readOnly" minEntries="1" maxEntries="1">
    <parameter name="Manufacturer" access="readOnly">
      …
    </parameter>
    …
  </object>
  …
</component>
```

I.7 `component` Element (reference)

This `component` element is used to reference an existing top-level component definition, and indicates that the content of the referenced component is to be included at the point of reference.

Possible parent elements include:

- `document/component` (I.6)
- `document/model` (I.8)
- `document/model/object` (I.9)

Table 16 lists the attributes that are available within this `component` element.

 © The Broadband Forum. All rights reserved

Table 16 – DM reference-based `component` attributes

Attribute	Type	Use	Description
ref	restricted xs:NCName	required	Name of an existing component definition to be referenced (included). **Note** – A `ref` has the same format as xs:NCName except that dots are not permitted; i.e. must start with a letter or "_" and subsequent characters can also include digits and separator characters such as underscore or dash.
path	restricted xs:NCName (256)	optional	If specified, it is the relative object path between point of reference (inclusion) and the component's items. If not specified, behavior is as if an empty relative object path was specified. Object path is represented by the concatenation of each successive object name separated by a dot (See Section 3.1/TR-106 [3]). **Note** – Each dot-separated portion of the `path` has the same format as xs:NCName except that dots are not permitted; i.e. must start with a letter or "_" and subsequent characters can also include digits and connector characters such as underscore and dash.

The following example uses a `component` reference in order to include the content of the existing DeviceInfo component (see example in previous section), which will be included at the specified `path` within the model. Since the referenced component defines object DeviceInfo., based on the `path` attribute the fully qualified name for this object becomes Device.DeviceInfo.; while the fully qualified name of the included DevieInfo.Manufacturer parameter becomes Device.DeviceInfo.Manufacturer.

```
<model name="Device:1.0">
  <component path="Device." ref="DeviceInfo"/>
  …
</model>
```

I.8 `model` Element

The top-level `model` element is used to define a new Root or Service model, either as an independent definition or as an extension to an existing model.

Possible parent elements include:

- `document` (I.1)

© The Broadband Forum. All rights reserved

Table 17 lists the attributes that are available within the `model` element.

Table 17 – DM `model` attributes

Attribute	Type	Use	Description
name	string(256)	required	The name of the model being defined, formatted as "<name>:<major-version>.<minor-version>". This value is unique within the document, including imported models. **Note** – The <name> part has the same format as xs:NCName except that dots are not permitted; i.e. must start with a letter or "_" and subsequent characters can also include digits and separator characters such as underscore or dash. The <major-version> and <minor-version> parts are digits.
base	string(256)	optional	The name of the model being extended, formatted as "<name>:<major-version>.<minor-version>" This attribute value must be present if and only if extending an existing model. See Section A.2.10/TR-106 [3].
isService	xs:boolean	optional	Indicates whether the model is a Service or Root model. The default is *false* if not specified (i.e. a Root model). **Note** – The top-level objects and parameters defined within a Service model are relative to (will appear under) a Root model's Services object.
status	One of: • current • deprecated • obsoleted • deleted	optional	The status of the model. The values *deprecated* and *obsoleted* are as described in Section 2.2/TR-106 [3]; *deleted* indicates the element is no longer present in the current Data Model; *current* indicates the element is present in the current Data Model. The default is *current* if not specified.
id	xs:token(256)	optional	Corresponds with the identifier from SMNP MIBs, included to allow easier import of these MIBs. Rarely used.

The following example defines a new Root model.

```
<model name="Device:2.0">
  …
</model>
```


© The Broadband Forum. All rights reserved

The following example defines a Root model that extends an existing model.

```
<model name="Device:1.1" base="Device:1.0">
  …
</model>
```

The following example defines a new Service model.

```
<model name="FAPService:1.0" isService="true">
  …
</model>
```

Table 18 lists the child elements allowed within a `model`. The `description` element if present must appear first, while the `profile` elements if present must appear last. The `component`, `parameter`, and `object` elements can appear in any order just so long as they come after any `description` and before any `profile` elements.

Table 18 – DM model sub-elements

Element	Multiplicity	Description
description	0 or 1	The model's description (I.2).
component	0 or more	A component reference (I.7).
parameter	0 or more	A top-level parameter definition (I.10).
object	0 or more	An object definition (I.9).
profile	0 or more	A profile definition (I.11).

I.9 object Element (definition)

This `object` element is used to define a new object, or to modify the definition of an existing object. Objects can be single-instance or multi-instance; the latter is often referred to as a table. It should not be confused with the `object` element that appears within profiles, which is an object reference rather than an object definition.

See Section A.2.8.1/TR-106 for details of how tables are represented.

Possible parent elements include:

- `document/model` (I.8)
- `document/component` (I.6)

Table 19 lists the attributes that are available within the `object` element.

Table 19 – DM definition-based object attributes

Attribute	Type	Use	Description
name	restricted xs:NCName	optional [29]	The name of a new object being defined, which includes its parent object path as

[29] An `object` element will contain a `name` attribute or a `base` attribute (one or the other). It is invalid for an

© The Broadband Forum. All rights reserved

	(256)		prefix (maximum length 256). This is represented by the concatenation of each successive parent object name separated by a dot (See Section 3.1/TR-106 [3]). **Note** – Each dot-separated portion of the overall name has the same format as xs:NCName except that dots are not permitted; i.e. must start with a letter or "_" and subsequent characters can also include digits and connector characters such as underscore and dash. Each object name is unique within its parent `model` or parent `component`. If the object is a table (see maxEntries), the final part of the name will be "`{i}.`" The name must in addition follow the vendor-specific object name requirements of Section 3.3/TR-106 [3]. This attribute is only used when defining a new object. When modifying an existing object, the `base` attribute is used instead.
base	restricted xs:NCName (256)	optional	The name of an existing object to be modified, which includes its parent object path. This attribute is only used when modifying an existing object. When defining a new object, the `name` attribute is used instead.
access	One of: • readOnly • readWrite	required	Whether object instances can be added or deleted by the ACS. Adding or deleting instances is meaningful only for a multi-instance (table) object.
minEntries	xs:nonNegativeInteger	required	The minimum number of instances of this object. minEntries must be less than or equal to maxEntries (all values are regarded as being less than "unbounded").
maxEntries	xs:positiveInte	required	The maximum number of instances of this

`object` element to omit both the `name` attribute and the `base` attribute.

 © The Broadband Forum. All rights reserved

	ger or "unbounded"		object. 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 :** object is a table
numEntriesParameter	string(256)	optional	The name of the parameter (in the parent object) that contains the number of entries in the table. This attribute must be specified for a table with a variable number of entries, i.e. for which maxEntries is greater than minEntries ("unbounded" is regarded as being greater than all values)
enableParameter	string(256)	optional	The name of the parameter (in each table entry) that enables/disables that table entry. This attribute must be specified for a table in which the ACS can create entries (i.e. access is readWrite) and which has one or more uniqueKey elements that define functional keys.
status	One of: • current • deprecated • obsoleted • deleted	optional	The status of the object. The values *deprecated* and *obsoleted* are as described in Section 2.2/TR-106 [3]; *deleted* indicates the element is no longer present in the current Data Model; *current* indicates the element is present in the current Data Model. The default is *current* if not specified. New object definitions are usually *current*, in which case this attribute can be omitted.
id	xs:token(256)	optional	Corresponds with the identifier from SMNP MIBs, included to allow easier

 © The Broadband Forum. All rights reserved

			import of these MIBs. Rarely used.

The following example defines a single-instance object; `maxEntries` is one.

```
<object name="Device.DeviceInfo." access="readOnly"
        minEntries="1" maxEntries="1">
  …
</object>
```

The following example defines a multi-instance object (i.e. a table); the object name ends in "{i}." and `maxEntries` is greater than one.

```
<object name="Device.Ethernet.Link.{i}." access="readWrite"
        numEntriesParameter="LinkNumberOfEntries" enableParameter="Enable"
        minEntries="0" maxEntries="unbounded">
  …
</object>
```

Table 20 lists the child elements allowed within an `object`. The order that these elements appear in the table is the same order, if present, that they must appear within an `object` definition (with the exception that `parameter` elements can appear before or after `component` elements).

Table 20 – DM definition-based `object` sub-elements

Element	Multiplicity	Description
description	0 or 1	The object's description (I.2).
uniqueKey	0 or more	Specifies the unique key for a table (I.9.1). This element is only permitted within table objects (see `maxEntries` in Table 19); i.e. not permitted within single-instance objects. The parameters referenced within each uniqueKey element together constitute a unique key. Keys can be functional or non-functional (I.9.1). 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.
component	0 or more	The components that are referenced (included) by the object (I.7).
parameter	0 or more	The object's parameter definitions (I.10).

 © The Broadband Forum. All rights reserved

I.9.1 Object `uniqueKey` Element

The `uniqueKey` element is used to specify the unique key for a multi-instance object (table). A table can have zero or more uniqueKeys, where each unique key consists of references to one or more of the table's child parameters.

Possible parent elements include:

- `document/[component|model]/object` (I.9)

Table 21 lists the attributes that are available within the object's `uniqueKey` element.

Table 21 – DM `uniqueKey` attributes

Attribute	Type	Use	Description
functional	xs:boolean	optional	The default is true if not specified. Indicates whether the key is a functional (true) or non-functional (false) key (see Section A.2.8.1/TR-106 [3]).

Table 22 lists the child elements allowed within an object's `uniqueKey`.

Table 22 – DM `uniqueKey` sub-elements

Element	Multiplicity	Description
parameter	1 or more	Reference to a parameter definition within the object (Table 23).

Table 23 lists the attributes that are available within the unique key's `parameter` element (i.e. attributes for element `uniqueKey/parameter`).

Table 23 – DM uniqueKey `parameter` attributes

Attribute	Type	Use	Description
ref	restricted xs:NCName (256)	required	The name of a parameter within the table to be included in the table's unique key. Note that the parameter's object path is not specified here. **Note** – A `ref` has the same format as xs:NCName except that dots are not permitted; i.e. must start with a letter or "_" and subsequent characters can also include digits and connector characters such as underscore and dash.

The following example shows a table with one unique key. This unique key consists of three of the table's parameters (i.e. a multi-parameter key). Also note that this is a functional key (by default, since the key's `functional` attribute is omitted), meaning it references at least one parameter that relates to the purpose (or function) of the table.

© The Broadband Forum. All rights reserved

```
<object name="Device.ManagementServer.ManageableDevice.{i}."
        minEntries="0" maxEntries="unbounded" … >
  <uniqueKey>
    <parameter ref="ManufacturerOUI"/>
    <parameter ref="SerialNumber"/>
    <parameter ref="ProductClass"/>
  </uniqueKey>
  …
</object>
```

The following example shows a table with two separate unique keys. In this case, each unique key consists of one of the table's parameters. Also, these are both defined as non-functional keys, meaning they reference parameters that do not relate to the purpose (or function) of the table.

```
<object name="Device.DSL.Channel.{i}."
        minEntries="0" maxEntries="unbounded" … >
  <uniqueKey functional="false">
    <parameter ref="Alias"/>
  </uniqueKey>
  <uniqueKey functional="false">
    <parameter ref="Name"/>
  </uniqueKey>
  …
</object>
```

I.10 `parameter` Element (definition)

This `parameter` element is used to define a new parameter, or to modify the definition of an existing parameter. It should not be confused with the `parameter` element that appears within profiles (I.11.2), which is a parameter reference rather than a parameter definition.

Possible parent elements include:

- `document/component` (I.6)
- `document/model` (I.8)
- `document/[component|model]/object` (I.9)

Table 24 lists the attributes that are available within the `parameter` element.

Table 24 – DM definition-based `parameter` attributes

Attribute	Type	Use	Description
name	restricted xs:NCName (256)	optional [30]	The name of the new parameter being defined (maximum length 256). Each parameter name is unique within its parent `model`, `component`, or `object`. Note that the parameter's

[30] A `parameter` element will contain a `name` attribute or a `base` attribute (one or the other). It is invalid for a `parameter` element to omit both the `name` attribute and the `base` attribute.

 © The Broadband Forum. All rights reserved

			object path is not specified here. **Note** – A `name` has the same format as xs:NCName except that dots are not permitted; i.e. must start with a letter or "_" and subsequent characters can also include digits and connector characters such as underscore and dash. This attribute is only used when defining a new parameter. When modifying an existing parameter, the `base` attribute is used instead. The name must in addition follow the vendor-specific object name requirements of Section 3.3/TR-106 [3].
base	restricted xs:NCName (256)	optional	The name of an existing parameter to be modified. Note that the parameter's object path is not specified here. This attribute is only used when modifying an existing parameter. When defining a new parameter, the `name` attribute is used instead.
access	One of: • readOnly • readWrite	required	Whether a parameter can be updated by the ACS.
status	One of: • current • deprecated • obsoleted • deleted	optional	The status of the parameter. The values *deprecated* and *obsoleted* are as described in Section 2.2/TR-106 [3]; *deleted* indicates the element is no longer present in the current Data Model; *current* indicates the element is present in the current Data Model. The default is *current* if not specified. New parameter definitions are usually *current*, in which case this attribute can be omitted.
activeNotify	One of: • normal • forceEnabled • forceDefaultEnabled • canDeny	optional	The default is *normal* if not specified.

 © The Broadband Forum. All rights reserved

forcedInform	xs:boolean	optional	The default is *false* if not specified.
id	xs:token(256)	optional	Corresponds with the identifier from SMNP MIBs, included to allow easier import of these MIBs. Rarely used.

The following example defines a writable parameter. The default values apply for omitted attributes (i.e. `status`=current, `activeNotify`=normal, `forcedInform`=false). Sub-elements not shown here would specify description and syntax.

```
<parameter name="PeriodicInformEnable" access="readWrite">
  …
</parameter>
```

The following example defines a read-only parameter. Optional attributes `activeNotify` and `forcedInform` are included since non-default values are desired.

```
<parameter name="ParameterKey" access="readOnly"
           activeNotify="canDeny" forcedInform="true">
  …
</parameter>
```

The following example illustrates the use of the `base` attribute. The existing parameter definition DeviceSummary is referenced in order to modify its status to *deprecated*.

```
<parameter base="DeviceSummary" access="readOnly" forcedInform="true"
           status="deprecated">
  …
</parameter>
```

Table 25 lists the child elements allowed within the `parameter`. The order that these elements appear in the table is the same order, if present, that they must appear within a `parameter` definition.

Table 25 – DM definition-based `parameter` sub-elements

Element	Multiplicity	Description
description	0 or 1	The parameter's description (I.2).
syntax	0 or 1	Contains the parameter's syntax definition (I.10.1). **Note** – For a new parameter (i.e. one defined using the `name` attribute rather than `base` attribute), the syntax element is required.

I.10.1 Parameter `syntax` Element

The `syntax` element is used to specify the parameter definition's data type details, where the data type is either one of the built-in types or is derived from a custom type defined elsewhere within the document.

 © The Broadband Forum. All rights reserved

Possible parent elements include:

- `document/[component|model]/parameter` (I.10)
- `document/[component|model]/object/parameter` (I.10)

Table 26 lists the attributes that are available within the parameter's `syntax` element.

Table 26 – DM parameter `syntax` attributes

Attribute	Type	Use	Description
hidden	xs:boolean	optional	The default is false if not specified. If true, will always read back as the null value for the parameter's base data type (see Section A.2.3.5/TR-106 [3]).
command	xs:boolean	optional	The default is false if not specified. If true, setting this parameter triggers a CPE action. Such a parameter is not part of the device configuration and will always read back as the null value for the parameter's base data type (see Section A.2.3.5/TR-106 [3]).

Table 27 lists the child elements allowed within a parameter's `syntax`. The `list` element if present must appear first, while the `default` element if present must appear last. One and only one of the built-in type elements, or the `dataType` element, can ever be present.

Table 27 – DM parameter `syntax` sub-elements

Element	Multiplicity	Description
list	0 or 1	Whether the parameter value is a list of items (I.10.2). For lists, the TR-069 parameter is always a string and the data type specification applies to individual list items, not to the parameter value.
base64 boolean dateTime hexBinary int long string unsignedInt unsignedLong	0 or 1[31]	Built-in primitive data type (I.12). Only one of these elements can be included within the `syntax`. If a built-in type is used, then the `dataType` element cannot be used.

[31] A `syntax` element can contain either one of the built-in type elements (i.e. boolean, string, int, etc) or a `dataType` element, or neither. For new parameter definitions, it is invalid for a `syntax` element not to include one such element.

© The Broadband Forum. All rights reserved

dataType	0 or 1	Reference to a named data type definition (I.10.3). If this element is used, then none of the built-in elements can be included within the syntax element.
default	0 or 1	The default value of the parameter (I.10.4).

The following example defines a parameter syntax that has a string type, using the string built-in data type element. Also, since hidden is true, read-back on such a parameter will always return an empty value. Syntax similar to this is often used with password parameters.

```
<syntax hidden="true">
  <string/>
</syntax>
```

The following example defines a parameter syntax that has a boolean type, using the boolean built-in data type element. Also, since command is true, setting such a parameter to true will trigger a CPE action.

```
<syntax command="true">
  <boolean/>
</syntax>
```

The following example defines a parameter syntax that has a MACAddress type, using the dataType element to reference the existing data type definition named MACAddress. Since the hidden attribute is omitted, hidden is false, and read back will return the actual value of the parameter.

```
<syntax>
  <dataType ref="MACAddress"/>
</syntax>
```

I.10.2 Syntax list Element

The list element indicates that the parameter's value will be a list of items. In this case the parameter is always a string type, and the data type specification instead applies to individual list items not to the parameter value.

Possible parent elements include:

- document/[component|model]/parameter/syntax (I.10.1)
- document/[component|model]/object/parameter/syntax (I.10.1)

Table 28 lists the attributes that are available within the syntax's list element.

Table 28 – DM syntax list attributes

Attribute	Type	Use	Description
minItems	xs:nonNegativeInteger	optional	The default is *0* if not specified.
maxItems	xs:positiveInteger or	optional	The default is *unbounded* if not specified.

 © The Broadband Forum. All rights reserved

	“unbounded”		
status	One of: • current • deprecated • obsoleted • deleted	optional	The status of the `list`. The values *deprecated* and *obsoleted* are as described in Section 2.2/TR-106 [3]; *deleted* indicates the element is no longer present in the current Data Model; *current* indicates the element is present in the current Data Model. The default is *current* if not specified. New `list` definitions are usually *current*, in which case this attribute can be omitted.

Table 29 lists the child elements allowed within a syntax’s `list`. The `description` element if present must appear first.

Table 29 – DM syntax `list` sub-elements

Element	Multiplicity	Description
description	0 or 1	The list description (I.2).
size	0 or more	The size of the list-valued parameter, not of the individual items (I.13.1).

The following example defines a parameter syntax having a list whose items are of type unsignedInt. This list can hold between 1 and 10 items.

```
<syntax>
  <list minItems=”1” maxItems=”10”/>
  <unsignedInt/>
</syntax>
```

The following example defines a parameter syntax that is a list of MACAddress type items. This list can hold between 0 and 16 items.

```
<syntax>
  <list maxItems="16"/>
  <dataType ref="MACAddress"/>
</syntax>
```

I.10.3 Syntax `dataType` Element (reference)

The syntax’s `dataType` element allows a parameter to be defined using a named data type (i.e. a custom type rather than one of the built-in types such as boolean, int, etc.). This is done by referencing a named data type definition which is specified elsewhere within the document. The referenced data type can be associated with a parameter definition as is, or it can be extended or restricted in order to provide the parameter with a modified variant of the referenced data type.

© The Broadband Forum. All rights reserved

Possible parent elements include:

- `document/[component|model]/parameter/syntax` (I.10.1)
- `document/[component|model]/object/parameter/syntax` (I.10.1)

Table 30 lists the attributes that are available within the syntax's `dataType` element.

Table 30 – DM syntax `dataType` attributes

Attribute	Type	Use	Description
ref	restricted xs:NCName	optional [32]	Reference to (the name of) an existing data type definition (I.4). When this attribute is specified, further content is not permitted within this `dataType` element (i.e. none of the facet sub-elements are permitted).
base	restricted xs:NCName	optional	Reference to (the name of) an existing data type definition (I.4) in order to define a new anonymous data type used by the parameter. The new data type can either be a restriction or extension of the named data type being referenced. When this attribute is specified, further content is required within this dataType element (i.e. one or more of the facet sub-elements will be used in order to alter the referenced definition).

Note – The data type name referenced in the `ref` and `base` attributes has the same format as xs:NCName except that it cannot start with a lower-case letter (to avoid conflict with built-in data type names) and dots are not permitted; i.e. must start with an upper-case letter or "_", and subsequent characters can also include digits and connector characters such as underscore and dash

Table 31 lists the child elements allowed within a syntax's `dataType` element.

Table 31 – DM syntax `dataType` sub-elements

Element	Multiplicity	Description
size instanceRef pathRef range enumeration enumerationRef pattern	0 or more	Data type facets (I.13). These are only permitted when the `base` attribute is used (i.e. when defining a new anonymous data type definition locally, based on an existing named data type definition that is being referenced).

[32] A syntax `dataType` element will contain a `ref` attribute or a `base` attribute (one or the other). It is invalid for this `dataType` element to omit both the `ref` attribute and the `base` attribute.

© The Broadband Forum. All rights reserved

units		

The following example defines a parameter syntax that has an IPAddress data type, as indicated by the `ref` attribute reference. The referenced data type is defined elsewhere within the document.

```
<syntax>
  <dataType ref="IPAddress"/>
</syntax>
```

The following example specifies a parameter syntax that defines an anonymous data type based on the referenced IPAddress type (as indicated by the `base` attribute). This new data type differs from the referenced type by using the `size` facet to restrict parameter values to a maximum length of 15.

```
<syntax>
  <dataType base="IPAddress">
    <size maxLength="15"/>
  </dataType>
</syntax>
```

I.10.4 Syntax `default` Element

The `default` element indicates the default value for a parameter. There are two types of defaults: factory default and object default.

Possible parent elements include:

- `document/[component|model]/parameter/syntax` (I.10.1)
- `document/[component|model]/object/parameter/syntax` (I.10.1)

Table 32 lists the attributes that are available within the syntax's `default` element.

Table 32 – DM syntax `default` attributes

Attribute	Type	Use	Description
type	One of: • factory • object	required	If *factory*, the `value` attribute specifies the parameter's default value based on some standard, e.g. RFC. This applies both to static parameters as well as parameters that come about due to object creation. If *object*, the `value` attribute specifies the parameter's default value. This only applies to parameters that come about due to object creation.
value	xs:string	required	The value must be valid for the data type.
status	One of:	optional	The status of the parameter's `default`

© The Broadband Forum. All rights reserved

	• current • deprecated • obsoleted • deleted		element. The values *deprecated* and *obsoleted* are as described in Section 2.2/TR-106 [3]; *deleted* indicates the element is no longer present in the current Data Model; *current* indicates the element is present in the current Data Model. The default value is *current* if not specified. New `default` element definitions are usually *current*, in which case this attribute can be omitted.

Table 33 lists the child elements allowed within the syntax's `default` element.

Table 33 – DM syntax `default` sub-elements

Element	Multiplicity	Description
description	0 or 1	The `default` element's description (I.2).

The following example defines the syntax for a boolean-valued parameter whose factory default value is *true*.

```
<syntax>
  <boolean/>
  <default type="factory" value="true"/>
</syntax>
```

The following example defines the syntax for an unsignedInt-valued parameter whose object default value is *0*. The optional `description` element provides further explanation about the default value.

```
<syntax>
  <unsignedInt/>
  <default type="object" value="0">
    <description>all bits clear</description>
  </default>
</syntax>
```

I.11 `profile` Element

The `profile` element is used to group together and specify access requirements for a subset of existing objects and parameters that are defined elsewhere within the Data Model. If a CPE supports a profile, it is expected that it will implement all of the object and parameter requirements defined within that profile. See Section A.2.9/TR-106 [3].

Possible parent elements include:

 © The Broadband Forum. All rights reserved

- document/component (I.6)
- document/model (I.8)

Table 34 lists the attributes that are available within the profile element.

Table 34 – DM profile attributes

Attribute	Type	Use	Description
name	restricted xs:NCName	optional[33]	The name of the profile being defined, formatted as "<name>:<version>". Each profile name is unique within its associated model. This attribute is only used when defining a new profile (or a new version of an existing profile). **Note** – The <name> part is the same as xs:NCName except that dots are not permitted; i.e. must start with a letter or "_" and subsequent characters can also include digits and connector characters such as underscore and dash. The <version> part contains digits only. For vendor-specific profiles, the naming convention discussed in Section 3.3/TR-106 [3] applies (i.e. in the form X_<VENDOR>_VendorSpecificName).
base	restricted xs:NCName	optional	The name of an existing profile to be modified. This attribute is only used when modifying an existing profile or the profile version is greater than 1. When defining a new profile that is not based on an existing profile, the name attribute is used and the base attribute is omitted.
extends	xs:list of profile names	optional	List of profile names. This is used when the profile extends other profile(s), and so is inheriting the other profiles' object and parameter requirements.
status	One of: • current • deprecated • obsoleted • deleted	optional	The status of the profile. The values *deprecated* and *obsoleted* are as described in Section 2.2/TR-106 [3]; *deleted* indicates the element is no longer present in the current Data Model; *current* indicates the element is present in the current Data Model. The default is *current* if not specified. New profile

[33] A profile element will contain a name attribute and/or a base attribute. It is invalid for a profile element to omit both the name attribute and the base attribute.

 © The Broadband Forum. All rights reserved

			definitions are usually *current*, in which case this attribute can be omitted.
id	xs:token(256)	optional	Corresponds with the identifier from SMNP MIBs, included to allow easier import of these MIBs. Rarely used.

The following example defines version 1 of the TempStatus profile. This is a new profile.

```
<profile name="TempStatus:1">
  …
</profile>
```

The following example defines version 1 of the TempStatusAdv profile. It is a new profile that extends the existing TempStatus:1 profile. It inherits requirements defined by TempStatus:1 without needing to specify them within TempStatusAdv:1. It is expected that TempStatusAdv:1 will define additional requirements not present within TempStatus:1.

```
<profile name="TempStatusAdv:1" extends="TempStatus:1">
  …
</profile>
```

The following example defines version 2 of the Baseline profile. Baseline:2 is an update to the previous Baseline:1 version. It is expected that Baseline:2 will only contain requirement changes, and that unchanged requirements will simply be inherited from Baseline:1.

```
<profile name="Baseline:2" base="Baseline:1">
  …
</profile>
```

The following example means to update the existing Baseline:1 profile directly, without defining a new profile or version. In effect, `name` is also Baseline:1. It is expected that only requirement changes appear within this profile definition update, and that unchanged requirements will simply be inherited from the existing Baseline:1 profile.

```
<profile base="Baseline:1">
  …
</profile>
```

Table 35 lists the child elements allowed within a `profile`. The `description` element if present must appear first. The `parameter` and `object` elements can appear in any order just so long as they do not come before a `description` element.

Table 35 – DM `profile` sub-elements

Element	Multiplicity	Description
description	0 or 1	The profile's description (I.2). If the `extends` attribute is insufficient to express general profile requirements, then any additional requirements will be

 © The Broadband Forum. All rights reserved

		specified here.
parameter	0 or more	A reference to an existing parameter definition (I.11.2). It includes requirements about this parameter.
object	0 or more	A reference to an existing object definition (I.11.1). It includes requirements about this object.

I.11.1 Profile `object` Element (reference)

The `profile` element can include zero or more `object` sub-elements. These are used to reference existing object definitions within the Data Model in order to specify their access requirements.

Possible parent elements include:

- `document/[component|model]/profile` (I.11)

Table 36 lists the attributes that are available within a profile's `object` element.

Table 36 – DM profile `object` attributes

Attribute	Type	Use	Description
ref	xs:NCName (256)	required	The name of an existing object, which includes its parent object path as prefix (maximum length 256). This is represented by the concatenation of each successive parent object name separated by a dot (See Section 3.1/TR-106 [3]). **Note** – Each dot-separated portion of the overall `name` has the same format as xs:NCName except that dots are not permitted; i.e. must start with a letter or "_" and subsequent characters can also include digits and connector characters such as underscore and dash.
requirement	One of: • notSpecified • present • create • delete • createDelete	required	Specifies the access requirement for the referenced object. Types of requirements are also discussed in Section 2.3.5/TR-106 [3]. **notSpecified**: No requirement specified for the object. The object is included within the profile simply to contain parameters. **present**: Indicates that the object must be present. For a multi-instance object, it cannot be created or deleted by the ACS. **create**, **delete**, **createDelete**: Used with a multi-instance object. Indicates whether the object can

© The Broadband Forum. All rights reserved

			have instances created and/or deleted.
status	One of: • current • deprecated • obsoleted • deleted	optional	The status of the `object` requirement. The values *deprecated* and *obsoleted* are as described in Section 2.2/TR-106 [3]; *deleted* indicates the element is no longer present in the current Data Model; *current* indicates the element is present in the current Data Model. The default is *current* if not specified. New object requirements are usually *current*, in which case this attribute can be omitted.

The following example defines the TempStatus:1 profile. It references the existing Device.TemperatureStatus object and indicates that this object must be present if a CPE supports this profile.

```
<profile name="TempStatus:1">
  <object ref="DeviceInfo.TemperatureStatus." requirement="present">
    …
  </object>
  …
</profile>
```

Table 37 lists the child elements allowed within a profile's `object` element. The `description` element if present must appear first.

Table 37 – DM profile `object` sub-elements

Element	Multiplicity	Description
description	0 or 1	The profile object's description (I.2). If the `requirement` attribute is insufficient to express the requirement, any additional requirements will be specified here and can override the attribute
parameter	0 or more	A reference to an existing parameter definition (I.11.2). It includes requirements about this parameter.

I.11.2 Profile `parameter` Element (reference)

The `profile` element can include zero or more `parameter` sub-elements, either directly under the `profile` element itself or within one of the profile's `object` elements. These are used to reference existing parameter definitions within the Data Model in order to specify their access requirements.

Possible parent elements include:

- `document/[component|model]/profile` (I.11)

© The Broadband Forum. All rights reserved

- `document/[component|model]/profile/object` (I.11.1)

Table 38 lists the attributes that are available within a profile's `parameter` element.

Table 38 – DM profile `parameter` attributes

Attribute	Type	Use	Description
ref	restricted xs:NCName (256)	required	The name of an existing parameter (maximum length 256). Note that the parameter's object path is not specified here. **Note** – A `name` has the same format as xs:NCName except that dots are not permitted; i.e. must start with a letter or "_" and subsequent characters can also include digits and connector characters such as underscore and dash.
requirement	One of: • readOnly • readWrite	required	Specifies the access requirement for the referenced parameter.
status	One of: • current • deprecated • obsoleted • deleted	optional	The status of the `parameter` requirement. The values *deprecated* and *obsoleted* are as described in Section 2.2/TR-106 [3]; *deleted* indicates the element is no longer present in the current Data Model; *current* indicates the element is present in the current Data Model. The default is *current* if not specified. New parameter requirements are usually *current*, in which case this attribute can be omitted.

The following example defines the User:1 profile. It illustrates how a parameter can be referenced (and requirements specified) both at the top-level of the profile, as well as from within `object` sub-elements.

```
<profile name="User:1">
  <parameter ref="UserNumberOfEntries" requirement="readOnly"/>

  <object ref="User.{i}." requirement="createDelete">
    <parameter ref="Enable" requirement="readWrite"/>
    …
  </object>
  …
</profile>
```

Table 39 lists the child elements allowed within a profile's `parameter` element. This applies whether the `parameter` appears directly under a `profile` element or under a profile's `object` element.

 © The Broadband Forum. All rights reserved

Table 39 – DM profile `parameter` sub-elements

Element	Multiplicity	Description
description	0 or 1	The profile parameter's description (I.2). If the `requirement` attribute is insufficient to express the requirement, any additional requirements will be specified here and can override the attribute.

I.12 Built-in Primitive Data Type Elements

The DM Schema comes equipped with a set of built-in primitive data types that can be used to define either top-level named data types or anonymous data types within parameters.

Possible parent elements include:

- `document/dataType` (I.4)
- `document/[component|model]/parameter/syntax` (I.10.1)
- `document/[component|model]/object/parameter/syntax` (I.10.1)

For any given parent element, no more than one primitive data type sub-element is allowed (e.g. a `dataType` element can include a `boolean` sub-element or a `string` sub-element, but not both).

Note that some primitive types can be specialized using certain data type facets, as indicated in the table below.

Table 40 lists the available primitive data type elements and their permitted facets.

Table 40 – DM primitive data type elements

Element	Description
base64	Base64 encoded binary (no line-length limitation). `base64` permits zero or more of the following sub-element facets: • `size` (Length is that of the actual string, not the base64-encoded string. See Section A.2.3.3/TR-106); see I.13.1
boolean	Boolean, where the allowed values are "true" (or "1"), and "false" (or "0").
dateTime	The subset of the ISO 8601 date-time format defined by the SOAP dateTime type.
hexBinary	Hex encoded binary. `hexBinary` permits zero or more of the following sub-element facets: • `size` (Length is that of the actual string, not the hexBinary-encoded string[34]. See Section A.2.3.3/TR-106); see I.13.1
int	Integer in the range -2147483648 to +2147483647, inclusive.

[34] In other words, it is the length of the actual string in bytes. Since a byte represents 2 hex digits, the length of the hexBinary-encoded value will be twice as many digits as the specified length.

© The Broadband Forum. All rights reserved

	`int` permits zero or more of the following sub-element facets: • `instanceRef` (I.13.2) • `range` (I.13.4) • `units` (I.13.8)
long	Long integer in the range –9223372036854775808 to 9223372036854775807, inclusive. `long` permits zero or more of the following sub-element facets: • `range` (I.13.4) • `units` (I.13.8)
string	Series of characters. `string` permits zero or more of the following sub-element facets: • `size` (I.13.1) • `pathRef` (I.13.3) • `enumeration` (Each enumeration value will be unique within the `string` element); see I.13.5 • `enumerationRef` (I.13.6) • `pattern` (Each pattern value will be unique within the `string` element); see I.13.7
unsignedInt	Unsigned integer in the range 0 to 4294967295, inclusive. `unsignedInt` permits zero or more of the following sub-element facets: • `instanceRef` (I.13.2) • `range` (I.13.4) • `units` (I.13.8)
unsignedLong	Unsigned long integer in the range 0 to 18446744073709551615, inclusive. `unsignedLong` permits zero or more of the following sub-element facets: • `range` (I.13.4) • `units` (I.13.8)

The following example lists all primitive types and their sub-element facets. Note that this is not a practical example, since these elements would actually appear independently within top-level dataType definitions or within parameter syntax.

```
<base64>
  <size …/>
</base64>

<boolean/>

<dateTime/>
```

 © The Broadband Forum. All rights reserved

```
<hexBinary>
  <size …/>
</hexBinary>

<int>
  <instanceRef …/>
  <range …/>
  <units …/>
</int>

<long>
  <range …/>
  <units …/>
</long>

<string>
  <size …/>
  <pathRef …/>
  <enumeration …/>
  <enumerationRef …/>
  <pattern …/>
</string>

<unsignedInt>
  <instanceRef …/>
  <range …/>
  <units …/>
</unsignedInt>

<unsignedLong>
  <range …/>
  <units …/>
</unsignedLong>
```

I.13 Data Type Facets

Facet elements can be used within data type and parameter definitions to specify different aspects of a data type being defined, such as size, units, etc.

Possible parent elements include:

- `document/dataType` (I.4)
- `document/[component|model]/parameter/syntax/dataType` (I.10.3)
- `document/[component|model]/object/parameter/syntax/dataType` (I.10.3)
- built-in primitive data types (I.12)

Note that some of the built-in primitive data types support a limited use of facets. This is discussed in Section I.12.

Table 41 lists the available facet elements. See Section A.2.3.3/TR-106 [3] for more details concerning these facet types.

© The Broadband Forum. All rights reserved

Note – For any given parent element, zero or more of each facet sub-element is permitted unless otherwise indicated.

Table 41 – DM data type facet elements

Element	Description
size	The minimum and maximum length for a string-related type (I.13.1). Multiple such elements can be used to indicate length ranges.
instanceRef	Reference to an object instance number; i.e. a row in a table (I.13.2).
pathRef	Reference to an object or parameter via its path name string (I.13.3).
range	The minimum and maximum value for an integer-related type (I.13.4). Multiple such elements can be used to define disjoint integer ranges.
enumeration	Specific value that is valid within a string-related type (I.13.5). Multiple such elements can be used to define a set of valid values.
enumerationRef	Reference to the enumeration values of another parameter via a path name string (I.13.6).
pattern	Pattern of valid values for a string-related type (I.13.7). Multiple such elements can be used to specify different patterns within a type definition.
units	Name of the units used for an integer-related type (I.13.8). Only one such element is permitted within a type definition.

Table 42 lists the child elements allowed within all of the facet elements.

Table 42 – DM facet sub-elements

Element	Multiplicity	Description
description	0 or 1	The facet's description (I.2).

I.13.1 `size` Element

Size facets, taken together, define the valid size ranges (i.e. string lengths), e.g. (0:0) and (6:6) mean that the size has to be 0 or 6. The `size` facet can only be specified for string data types, i.e. data types that are derived from base64, hexBinary or string.

Table 43 lists the attributes that are available within a `size` element.

Table 43 – DM `size` attributes

Attribute	Type	Use	Description
access	One of: • readOnly • readWrite	optional	The default is *readWrite* if not specified.
minLength	xs:nonNegativeInteger	optional	The default is *0* if not specified.
maxLength	xs:nonNegativeInteger	optional	The default is *16* if not specified and no implied maximum exists (see Section 3.2.6/TR-106 [3] for further details).
optional	xs:boolean	optional	The default is *false* if not specified.
status	One of:	optional	The status of the *size* element.

 © The Broadband Forum. All rights reserved

	• current • deprecated • obsoleted • deleted		The values *deprecated* and *obsoleted* are as described in Section 2.2/TR-106 [3]; *deleted* indicates the element is no longer present in the current Data Model; *current* indicates the element is present in the current Data Model. The default is *current* if not specified. New elements are usually *current*, in which case this attribute can be omitted.

Note – The `access` and `optional` attributes serve no purpose. They should not be used and may be removed in a future version of the DM Schema.

The following example defines a string length size between 1 and 255 characters.

```
<size minLength="1" maxLength="255"/>
```

I.13.2 `instanceRef` Element

InstanceRef facets specify how a parameter can reference an object instance (table row) via its instance number. The `instanceRef` facet can only be specified for data types that are derived from int or unsignedInt.

Table 44 lists the attributes that are available within an `instanceRef` element.

Table 44 – DM `instanceRef` attributes

Attribute	Type	Use	Description
targetParent	Path Name	required	Specifies the path name of the multi-instance object (table) of which an instance (row) is to be referenced. Specified path must reference a specific multi-instance object (table) (Section A.2.3.4/TR-106). Object path name cannot contain "{i}" placeholders and therefore will reference a single object. Path name cannot contain explicit instance numbers. The path name will follow the requirements of Section A.2.3.4/TR-106 (and with scope specified via the `targetParentScope` attribute).
targetParentScope	One of:	optional	Object path name scope. Specifies the

 © The Broadband Forum. All rights reserved

	• normal • model • object		point in the naming hierarchy relative to which targetParent applies (Section A.2.3.4/TR-106). **normal**: This is a hybrid scope which usually gives the desired behavior. If the `targetParent` path begins with "Device" or "InternetGatewayDevice" then it is relative to the top of the naming hierarchy. If the `targetParent` path begins with a dot then it is relative to the Root or Service Object. Otherwise, the `targetParent` path is relative to the current object. **model**: The `targetParent` path is relative to the Root or Service Object. **object**: The `targetParent` path is relative to the current object. The default is *normal* if not specified. **Note** – The preference is to omit this attribute and to set `targetParent` with a value that is formatted to imply a path scope.
refType	One of: • weak • strong	required	Specifies the type of reference (Section A.2.3.6/TR-106). If *weak*, the referenced object instance might not exist (e.g. referenced object was deleted by ACS). If *strong*, the referenced object instance will exist, otherwise the CPE will set the parameter to a null reference (see Section A.2.3.5/TR-106).
status	One of: • current • deprecated • obsoleted • deleted	optional	The status of the `instanceRef` element. The values *deprecated* and *obsoleted* are as described in Section 2.2/TR-106; *deleted* indicates the element is no longer present in the current Data Model; *current* indicates the element is present in the current Data Model.

 © The Broadband Forum. All rights reserved

			The default is *current* if not specified. New elements are usually *current*, in which case this attribute can be omitted.

The following example defines an `instanceRef` which will strongly reference a row in the Queue table.

```
<instanceRef refType="strong" targetParent="Device.QueueManagement.Queue."/>
```

The following example defines an `instanceRef` which will weakly reference a row in the Diagnostics table. The `targetParentScope` attribute indicates that the path scope is the current object (e.g. if the current parameter were Device.SomeObject.SomeParameter, then it is expected that the target parent is Device.SomeObject.Diagnostics).

```
<instanceRef refType="weak" targetParent="Diagnostics."
             targetParentScope="object"/>
```

I.13.3 `pathRef` Element

PathRef facets specify how a parameter can reference another parameter or object via a path name. The `pathRef` facet can only be specified for data types that are derived from string (i.e. string and its derived types).

Table 45 lists the attributes that are available within a `pathRef` element.

Table 45 – DM `pathRef` attributes

Attribute	Type	Use	Description
targetParent	list of Path Names	optional	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, and can therefore reference multiple objects. Path names cannot contain explicit instance numbers. Each path name will follow the requirements of Section A.2.3.4/TR-106 (with path name scope specified by the `targetParentScope` attribute).

 © The Broadband Forum. All rights reserved

targetParentScope	One of: • normal • model • object	optional	Object/parameter path name scope. Specifies the point in the naming hierarchy relative to which `targetParent` applies (see Section A.2.3.4/TR-106). **normal**: This is a hybrid scope which usually gives the desired behavior. If the `targetParent` path(s) begins with "Device" or "InternetGatewayDevice" then it is relative to the top of the naming hierarchy. If the `targetParent` path(s) begins with a dot then it is relative to the Root or Service Object. Otherwise, the `targetParent` path(s) is relative to the current object. **model**: The `targetParent` path(s) is relative to the Root or Service Object. **object**: The `targetParent` path(s) is relative to the current object. The default is *normal* if not specified. **Note** – The preference is to omit this attribute and to set `targetParent` with a value that is formatted to imply a path scope.
targetType	One of: • any • parameter • object • single • table • row	optional	Specifies the type of item that can be referenced by `targetParent` (see Section A.2.3.7/TR-106). **any**: Either a parameter or an object can be referenced. **parameter**: Only a parameter can be referenced. **object**: Any type of object can be referenced. **single**: Only a single-instance object can be referenced. **table**: Only a multi-instance object

 © The Broadband Forum. All rights reserved

			(table) can be referenced. **row**: Only a multi-instance object instance (table row) can be referenced. The default is *any* if not specified.
targetDataType	One of: • any • base64 • boolean • dateTime • hexBinary • integer • int • long • string • unsignedInt • unsignedLong • <named data type>	optional	Specifies the valid data types for a referenced parameter (see Section A.2.3.7/TR-106). Is relevant only when `targetType` is *any* or *parameter*. The default is *any* if not specified. For named data types, see Section I.4. For primitive data types, see Section I.12. 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	One of: • weak • strong	required	Specifies the type of reference (Section A.2.3.6/TR-106). If *weak*, the referenced parameter or object might not exist (e.g. referenced object was deleted by ACS). If *strong*, the referenced parameter or object will exist, otherwise the CPE will set the parameter to a null reference (see Section A.2.3.5/TR-106).
status	One of: • current • deprecated • obsoleted • deleted	optional	The status of the `pathRef` element. The values *deprecated* and *obsoleted* are as described in Section 2.2/TR-106; *deleted* indicates the element is no longer present in the current Data Model; *current* indicates the element is present in the current Data Model. The default is *current* if not specified. New elements are usually *current*, in which case this attribute can be omitted.

© The Broadband Forum. All rights reserved

The following example defines a `pathRef` which can reference any parameter. This is a weak reference.

```
<pathRef refType="weak" targetType="parameter"/>
```

The following example defines a `pathRef` which can only reference a boolean-typed parameter. This example is similar to the previous except for the addition of the `targetDataType` attribute.

```
<pathRef refType="weak" targetType="parameter"
         targetDataType="boolean"/>
```

The following example defines a `pathRef` which can only reference an object instance (row) within the Bridge or VLAN tables. This is a strong reference. Note that the `targetParentScope` attribute could have been omitted, since the targetParent's path names start with a dot and therefore already imply that they are relative to the Root (or Service) Object.

```
<pathRef refType="strong"
         targetParent=".Bridging.Bridge. .Bridging.Bridge.{i}.VLAN."
         targetParentScope="model"
         targetType="row"/>
```

The following example defines a `pathRef` which can only reference a row in the Router table. This example differs from the previous in that the `targetParentScope` attribute has been omitted; however, since the `targetParent` attribute value starts with a dot, the target path name is relative to the Root Object by default.

```
<pathRef refType="strong"
         targetParent=".Routing.Router."
         targetType="row"/>
```

The following example defines a `pathRef` which can only reference a row within the Profile table. Since the `targetParentScope` attribute is omitted, and the `targetParent` attribute value does not start with "Device" or "InternetGatewayDevice" or a dot, the target path name is relative to the current object by default.

```
<pathRef refType="strong"
         targetParent="Profile."
         targetType="row"/>
```

I.13.4 `range` Element

Range facets, taken together, define the valid value ranges, e.g. [-1:-1] and [1:4094] mean that the value has to be -1 or 1:4094 (it cannot be 0). The `range` facet can only be specified for numeric data types, i.e. data types that are derived from one of the integer types.

Table 46 lists the attributes that are available within a `range` element.

 © The Broadband Forum. All rights reserved

Table 46 – DM `range` attributes

Attribute	Type	Use	Description
access	One of: • readOnly • readWrite	optional	Whether values within the specified range can be can be written by the ACS. The default is *readWrite* if not specified.
minInclusive	xs:integer	optional	Minimum value in the range. If omitted, the default minimum is the minimum allowed by the base type.
maxInclusive	xs:integer	optional	Maximum value in the range. If omitted, the default maximum is the maximum allowed by the base type.
step	xs:positiveInteger	optional	The default is *1* if not specified.
optional	xs:boolean	optional	Whether values within the specified range are optionally supported by the CPE. The default is *false* if not specified.
status	One of: • current • deprecated • obsoleted • deleted	optional	The status of the *range* element. The values *deprecated* and *obsoleted* are as described in Section 2.2/TR-106; *deleted* indicates the element is no longer present in the current Data Model; *current* indicates the element is present in the current Data Model. The default is *current* if not specified. New elements are usually *current*, in which case this attribute can be omitted.

The following example defines a range where the parameter value can be -1 or between 1 and 10. Zero is not a valid value. Also, the ACS can only write values between 1 and 10.

```
<range access="readOnly" minInclusive="-1" maxInclusive="-1">
<range minInclusive="1" maxInclusive="10">
```

The following example defines a range where the parameter value can be 3, 6, 9, or 12; i.e. valid values step by 3 starting from the `minInclusive` value.

```
<range minInclusive="3" maxInclusive="12" step="3">
```

I.13.5 `enumeration` Element

Enumeration facets, taken together, define the valid values, e.g. "a" and "b" mean that the value has to be a or b. The `enumeration` facet can only be specified for data types that are derived from string. Derived types may add additional enumeration values. See Section A.2.5/TR-106.

© The Broadband Forum. All rights reserved

Table 47 lists the attributes that are available within an `enumeration` element.

Table 47 – DM `enumeration` attributes

Attribute	Type	Use	Description
access	One of: • readOnly • readWrite	optional	Whether an enumeration value can be written by the ACS. The default is *readWrite* if not specified.
value	xs:string	required	An enumeration value. Duplicate values are not allowed within the associated parameter.
code	xs:integer	optional	An enumeration numeric code.
optional	xs:boolean	optional	Whether an enumeration value is optionally supported by the CPE. The default is *false* if not specified.
status	One of: • current • deprecated • obsoleted • deleted	optional	The status of the `enumeration` element. The values *deprecated* and *obsoleted* are as described in Section 2.2/TR-106; *deleted* indicates the element is no longer present in the current Data Model; *current* indicates the element is present in the current Data Model. The default is *current* if not specified. New elements are usually *current*, in which case this attribute can be omitted.

Note that the maximum string length of a data type defined with enumeration facets is implied by the length of its longest enumeration value.

The following example defines the set of enumeration values: None, Requested, Complete, and Error. Each enumeration value is unique to the set. Only the Requested value can be written by the ACS to the associated parameter. The Error value is optionally supported by the CPE.

```
<enumeration value="None" access="readOnly"/>
<enumeration value="Requested"/>
<enumeration value="Complete" access="readOnly"/>
<enumeration value="Error" access="readOnly" optional="true"/>
```

The following example builds on the previous example. Here numeric codes are indicated for each enumeration.

```
<enumeration value="None" code="0" access="readOnly"/>
<enumeration value="Requested" code="1"/>
<enumeration value="Complete" code="2" access="readOnly"/>
<enumeration value="Error" code="3" access="readOnly"/>
```


© The Broadband Forum. All rights reserved

I.13.6 enumerationRef Element

EnumerationRef facets allow a parameter's valid enumeration values to be obtained from the current value of another parameter (by referencing a list-valued parameter via a path name). The enumerationRef facet can only be specified for string types and those types derived from string.

Table 48 lists the attributes that are available within an enumerationRef element.

Table 48 – DM enumerationRef attributes

Attribute	Type	Use	Description
targetParam	Path Name	required	Specifies the path name of the list-valued parameter whose current value indicates the valid enumerations for this parameter. Parameter path cannot contain "{i}" placeholders and therefore will reference a single parameter. The path must follow the requirements of Section A.2.3.4/TR-106 (its path scope is specified by the targetParamScope attribute).
targetParamScope	One of: • normal • model • object	optional	Parameter path name scope. Specifies the point in the naming hierarchy relative to which targetParam applies (see Section A.2.3.4/TR-106). **normal**: This is a hybrid scope which usually gives the desired behavior. If the targetParam path begins with "Device" or "InternetGatewayDevice" then it is relative to the top of the naming hierarchy. If the targetParam path begins with a dot then it is relative to the Root or Service Object. Otherwise, the targetParam path is relative to the current object. **model**: The targetParam path is relative to the Root or Service Object. **object**: The targetParam path is relative to the current object. The default is *normal* if not specified.

© The Broadband Forum. All rights reserved

			Note – The preference is to omit this attribute and to set `targetParam` with a value that is formatted to imply a path scope.
nullValue	xs:token	optional	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** – This attribute is not relevant when the data type is list-valued, because the null value will be indicated by an empty list.
status	One of: • current • deprecated • obsoleted • deleted	optional	The status of the `enumerationRef` element. The values *deprecated* and *obsoleted* are as described in Section 2.2/TR-106; *deleted* indicates the element is no longer present in the current Data Model; *current* indicates the element is present in the current Data Model. The default is *current* if not specified. New elements are usually *current*, in which case this attribute can be omitted.

The following example defines an `enumerationRef` which references the AllowedProfiles list-valued parameter. The null value is empty string. Note that the `targetParamScope` attribute could have been omitted, since the `targetParam` value starts with a dot and therefore already implies that it is relative to the Root (or Service) Object.

```
<enumerationRef targetParam=".SomeObject.AllowedProfiles"
                targetParamScope="model"
                nullValue=""/>
```

The following example defines an `enumerationRef` which references the StandardsSupported list-valued parameter. This example differs from the previous in that the `targetParamScope` attribute has been omitted; however, since the `targetParam` attribute value does not start with "Device" or "InternetGatewayDevice" or a dot, the target path name is relative to the current object by default.

```
<enumerationRef targetParam="StandardsSupported" nullValue=""/>
```


© The Broadband Forum. All rights reserved

I.13.7 `pattern` Element

Pattern attributes, taken together, define valid patterns, e.g. "" and "[0-9A-Fa-f]{6}" means that the value has to be empty or a 6 digit hex string. The `pattern` facet can only be specified for data types that are derived from string.

Note – The pattern syntax is the same as for XML Schema regular expressions. See XML Schema Part 2 [12] Appendix F.

Table 49 lists the attributes that are available within a `pattern` element.

Table 49 – DM `pattern` attributes

Attribute	Type	Use	Description
access	One of: • readOnly • readWrite	optional	Whether a parameter value that matches the pattern value can be written by the ACS. The default is *readWrite* if not specified.
value	xs:string	required	Pattern for the data type (a regular expression).
optional	xs:boolean	optional	Whether a pattern value is optionally supported by the CPE. The default is *false* if not specified.
status	One of: • current • deprecated • obsoleted • deleted	optional	The status of the `pattern` element. The values *deprecated* and *obsoleted* are as described in Section 2.2/TR-106; *deleted* indicates the element is no longer present in the current Data Model; *current* indicates the element is present in the current Data Model. The default is *current* if not specified. New elements are usually *current*, in which case this attribute can be omitted.

The following example defines a set of five patterns for some data type or parameter. The first four patterns are constant values. The last pattern defines a value which includes an “X”, a space, any six digit hex number, a space, followed by any sequence of characters.

```
<pattern value="1 Firmware Upgrade Image"/>
<pattern value="2 Web Content"/>
<pattern value="3 Vendor Configuration File"/>
<pattern value="4 Vendor Log File"/>
<pattern value="X [0-9A-F]{6} .*"/>
```

 © The Broadband Forum. All rights reserved

I.13.8 `units` Element

Multiple units facets must not be specified. The `units` facet can only be specified for data types that are numeric, i.e. data types that are derived from one of the integer types.

Table 50 lists the attributes that are available within a `units` element.

Table 50 – DM `units` attributes

Attribute	Type	Use	Description
value	xs:token (32)	required	The name of the units (maximum length of 32).
status	One of: • current • deprecated • obsoleted • deleted	optional	The status of the `units` element. The values *deprecated* and *obsoleted* are as described in Section 2.2/TR-106; *deleted* indicates the element is no longer present in the current Data Model; *current* indicates the element is present in the current Data Model. The default is *current* if not specified. New elements are usually *current*, in which case this attribute can be omitted.

The following example defines the units for some data type or parameter to be percent.

```
<units value="percent"/>
```

 © The Broadband Forum. All rights reserved

Appendix II – Reference: Device Type XML Schema

This appendix provides a user reference for the TR-069 device type schema (DT Schema), v1.1. The normative version can be found at http://www.broadband-forum.org/cwmp/cwmp-devicetype-1-1.xsd.

The DT Schema is used to define DT Instance documents. For a given device type, a DT Instance document specifies which Data Model items (e.g. objects, parameters, parameter values) from an associated DM Instance document will be supported by devices of that type.

II.1 `document` Element

The `document` element is the root element of any DT XML Data Model file. It is required.

Table 51 lists the attributes that are available within the `document` element.

Table 51 – DT `document` attributes

Attribute	Type	Use	Description
deviceType	xs:anyURI	required	URI indicating the device type associated with this DT Instance. An empty string is not allowed. This URI is a globally uniquely identifier; it identifies the device type not the document.

The following example illustrates the use of the `document` element. It indicates the `deviceType` that the `document` is associated with and also specifies a `dt` namespace to be used throughout the document.

```
<dt:document xmlns:dt="urn:broadband-forum-org:cwmp:devicetype-1-1"
             deviceType="urn:your-company-com:example">
  …
</dt:document>
```

Note – In the above example, although outside the scope of this appendix, it is expected that the `document` element will also include the `xmlns:xsi` and `xsi:schemaLocation` attributes in order to specify the location of the XSD schema file to be associated with the `dt` namespace.

Table 52 lists the child elements allowed within a `document`. The order that these elements appear in the table is the same order, if present, that they must appear within a `document` definition.

Table 52 – DT `document` sub-elements

Element	Multiplicity	Description
annotation	0 or 1	Top-level annotation (II.2).

© The Broadband Forum. All rights reserved

import	0 or more	Imported model (Root and Service Objects) and data type definitions (II.4). The imported items are defined in DM Instance documents.
bibliography	0 or 1	Bibliographic references (II.5).
feature	0 or more	Top-level features (II.3). Declares which features of imported Root and Service Objects are supported.
model	0 or more	Shows support for a model (Root and Service Object) (II.6).

II.2 `annotation` Element

The `annotation` element holds free text (i.e. of type xs:string) used to describe further details regarding how the parent element is supported by the device type. This text can contain a limited amount of Media Wiki-like markup as specified in Sections I.2.2 and I.2.3. For example, use "*" at the start of a line to indicate a bulleted list. To avoid confusion, the annotation should not contain tab characters.

For Broadband Forum standards, the character set will 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.

The annotation element can be used within almost every other element (e.g. under `document`, `document/model`, `document/model/object`, etc.). When used, it will always be the first element to appear under its parent element. Refer to specific sections within this appendix to determine whether or not the annotation element is permitted.

The following example illustrates the structure of an annotation.

```
<annotation>
  This text describes the parent element…
</annotation>
```

II.3 `feature` Element

The `feature` element is used to declare which features of imported models (Root and Service Objects) are supported. These are local declarations of purported named features and are not validated against actual definitions from imported DM Instance models.

The specific list of feature names that can be referenced is specified by the separate Device Type Features (DTF) XML Schema. However, this is transparent to users of the DT Schema. See Appendix III for DTF Schema details, and Section B.2.2/TR-106 [3] which also discusses features.

Possible parent elements include:

- `document` (II.1)

 © The Broadband Forum. All rights reserved

Table 53 lists the attributes that are available within the `feature` element.

Table 53 – DT `feature` attributes

Attribute	Type	Use	Description
name	One of: • DNSClient • DNSServer • Firewall • IPv6 • NAT • Router	required	Feature name. The schema also allows vendors to use their own feature names. Vendor-specific feature names will begin with X_<VENDOR>_, where <VENDOR> must be as defined in Section 3.3/TR-106 [3] (e.g. X_ACDC73_NetworkStorage).

The following example illustrates the use of a `feature` element.

```
<feature name="Firewall"/>
```

Table 54 lists the child elements allowed within an `import`. These are explained in greater detail in the next section.

Table 54 – DT `feature` sub-elements

Element	Multiplicity	Description
annotation	0 or 1	The feature's annotation (II.2).

II.4 `import` Element

The `import` element is used to import data types and models (Root and Service Objects) from external DM Instance documents.

By importing such item definitions, they are then available to be referenced throughout the local document. However, if these imported items are not subsequently referenced, then they do not actually contribute to the local document's Data Model (i.e. importing items from an external file does not automatically make them part of the model defined within the local document).

The import mechanism is recursive; i.e. if an imported file itself includes imports, then these imports are also available in the local document (as is the case with the imported file's imported file's, and so on).

Possible parent elements include:

- `document` (II.1)

Table 55 lists the attributes that are available within the `import` element.

Table 55 – DT `import` attributes

Attribute	Type	Use	Description

 © The Broadband Forum. All rights reserved

file	xs:anyURI	required	URI of the file. It must be used to locate the DM Instance (see Section B.2.1/TR-106).
spec	xs:anyURI	optional	URI of the spec. If specified, processing tools must regard a mismatch between this and the external document's spec attribute as an error.

The following example illustrates how to reference an external DM Instance document. Specific items to be imported from this file are specified using `import` sub-elements.

```
<import file="tr-143-1-0.xml" spec="urn:broadband-forum-org:tr-143-1-0">
  …
</import>
```

Table 56 lists the child elements allowed within an `import`. These are explained in greater detail in the next section. Note that `dataType` and `model` elements can appear in any order, but the convention is to list `dataType` elements first.

Table 56 – DT `import` sub-elements

Element	Multiplicity	Description
dataType	0 or more	Reference to a `dataType` in the external document (II.4.1).
model	0 or more	Reference to a `model` in the external document (II.4.1).

II.4.1 `import` Sub-elements

The `import` element has two possible child elements: `dataType` and `model`. Each is used to indicate specific items to be imported from the external document specified by the parent element.

Possible parent elements include:

- `document/import` (II.4)

Table 57 lists the attributes that are available within the `dataType` and `model` sub-elements.

Table 57 – DT `import` sub-element attributes

Attribute	Type	Use	Description
name	restricted xs:NCName	required	Name of the item being imported from the external document. Note that the item will also be referenced by this same name within the local document.

The following example imports the IPAddress data type from tr-106-1-0-types.xml.

```
<import file="tr-106-1-0-types.xml" spec="urn:broadband-forum-org:tr-106-1-0">
  <dataType name="IPAddress"/>
  …
</import>
```


© The Broadband Forum. All rights reserved

Note – The format of the `dataType/@name` attribute is the same as xs:NCName, except that it cannot start with a lower-case letter (to avoid conflict with built-in data type names) and dots are not permitted. However, this is academic really, since the name must match a corresponding data type already defined in the associated DM Instance document.

The following example imports the Device:1.3 model from tr-157-1-0.xml.

```
<import file="tr-157-1-0.xml" spec="urn:broadband-forum-org:tr-157-1-0">
  <model name="Device:1.3"/>
  …
</import>
```

Note – The `model/@name` attribute value includes a name part, a colon, major version digits, a dot, and minor version digits. The format of the name part is the same as xs:NCName, except that dots are not permitted. However, this is academic really, since the name and version must match a corresponding model already defined in the associated DM Instance document.

Even though it is not shown here, an `import` example could be crafted that imports data types and models from the same external document.

II.5 bibliography Element

The `bibliography` element is used to define bibliographic references for various publications that might be cited throughout the document. This is the very same as the DM Schema's `bibliography` element (I.5).

Possible parent elements include:

- `document` (II.1)

Table 58 lists the elements allowed within a `bibliography`. The order that these elements appear in the table is the same order, if present, that they must appear within a `bibliography` definition.

Table 58 – DT bibliography sub-elements

Element	Multiplicity	Description
description	0 or 1	The bibliography's description (I.2). **Note** – This element will be deprecated in DT Schema v1.2.
annotation	0 or 1	The bibliography's annotation (II.2). **Note** – This element will be defined in DT Schema v1.2, as a replacement for the `description` element.
reference	0 or more	Information about the referenced publication; i.e. name, title, date, hyperlink, etc. (II.5.1).

 © The Broadband Forum. All rights reserved

II.5.1 Bibliography `reference` Element

The `bibliography` element's `reference` sub-element is used to specify the details regarding a particular publication. A publication can be referenced from annotations throughout the document using the `{{bibref}}` Template (see I.2.3).

Possible parent elements include:

- `document/bibliography` (II.5)

Table 59 lists the attributes that are available within the `reference` element.

Table 59 – DT `bibliography reference` attributes

Attribute	Type	Use	Description
id	xs:token	required	Uniquely identifies the bibliographic reference within the document (and should uniquely identify this reference across all instance documents). For BBF DM Instances, the bibliographic reference ID rules specified in Section A.2.4/TR-106 must be used. For example, to reference TR-106 Issue 1 Amendment 2, the value of this attribute would be TR-106a2.

Table 60 lists the elements allowed within a `reference`. The order that these elements appear in the table is the same order, if present, that they must appear within a `reference` definition.

Table 60 – DT `bibliography reference` sub-elements

Element	Multiplicity	Type	Description
name	1	xs:token	Name by which the referenced document is usually known, e.g. TR-069, RFC 2863. This is a required element.
title	0 or 1	xs:token	Title of the referenced document.
organization	0 or 1	xs:token	Organization that published the referenced document, e.g. BBF, IEEE, IETF.
category	0 or 1	xs:token	Document category, e.g. TR (BBF), RFC (IETF).
date	0 or 1	xs:token	Publication date.
hyperlink	0 or more	xs:anyURI	URI where the referenced document can be found.

The following example illustrates a bibliographic reference for RFC 2863.

```
<bibliography>
  <reference id="RFC2863">
    <name>RFC 2863</name>
    <title>The Interfaces Group MIB</title>
    <organization>IETF</organization>
    <category>RFC</category>
    <date>June 2000</date>
    <hyperlink>http://tools.ietf.org/html/rfc2863</hyperlink>
```


© The Broadband Forum. All rights reserved

```
  </reference>
  …
</bibliography>
```

II.6 `model` Element

The top-level `model` element is used to specify support for a Root or Service model defined in a DM Instance.

Possible parent elements include:

- `document` (II.1)

Table 61 lists the attributes that are available within the `model` element.

Table 61 – DT `model` attributes

Attribute	Type	Use	Description
ref	string(256)	required	The name of the model being referenced, formatted as “<name>:<major-version>.<minor-version>”. **Note** – The <name> part has the same format as xs:NCName except that dots are not permitted; i.e. must start with a letter or “_” and subsequent characters can also include digits and separator characters such as underscore or dash. The <major-version> and <minor-version> parts are digits.

The following example illustrates support for the Device:2.0 Data Model.

```
<model ref="Device:2.0">
  …
</model>
```

Table 62 lists the child elements allowed within a `model`. The `annotation` element if present must appear first. The `parameter` and `object` elements can appear in any order just so long as they come after any `annotation` (but the convention is to list top-level parameter elements before object elements).

Table 62 – DT `model` sub-elements

Element	Multiplicity	Description
annotation	0 or 1	The model’s annotation (II.2).
parameter	0 or more	Shows support for a top-level parameter (II.8).
object	0 or more	Shows support for an object (II.7).

© The Broadband Forum. All rights reserved

II.7 `object` Element

The `object` element is used to specify support for an object defined in a DM Instance.

Possible parent elements include:

- `document/model` (II.6)

Table 63 lists the attributes that are available within the `object` element.

Table 63 – DT `object` attributes

Attribute	Type	Use	Description
ref	restricted xs:NCName (256)	required	The name of the object being referenced, which includes its parent object path as prefix (maximum length 256). This is represented by the concatenation of each successive parent object name separated by a dot (See Section 3.1/TR-106 [3]). **Note** – Each dot-separated portion of the overall name has the same format as xs:NCName except that dots are not permitted; i.e. must start with a letter or "_" and subsequent characters can also include digits and connector characters such as underscore and dash. Each object name is unique within its parent `model`. If the object is a table (see maxEntries), the final part of the name will be "{i}." The name must in addition follow the vendor-specific object name requirements of Section 3.3/TR-106 [3].
access	One of: • readOnly • create • delete • createDelete	required	Whether object instances can be added or deleted by the ACS. Adding or deleting instances is meaningful only for a multi-instance (table) object.
minEntries	xs:nonNegativeInteger	required	The minimum number of instances supported of this object. minEntries must be less than or equal to maxEntries (all values are regarded as being less than "unbounded").

© The Broadband Forum. All rights reserved

			Note that this minEntries must be greater than or equal to the minEntries defined in the Data Model (DM) for the object.
maxEntries	xs:positiveInteger or "unbounded"	required	The maximum number of instances supported of this object. All numeric values are regarded as being less than "unbounded". Note that this maxEntries must be less than or equal to the maxEntries defined in the Data Model (DM) for the object. minEntries and maxEntries indicate the category of object being referenced: • **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 :** object is a table

The following example illustrates support for the DeviceInfo object (a single-instance object). The specified attribute values are the same as in the DM definition but are re-specified here since they are required attributes.

```
<object ref="Device.DeviceInfo." access="readOnly"
        minEntries="1" maxEntries="1">
  …
</object>
```

The following example illustrates support for the Ethernet Link object (a multi-instance object; i.e. a table). Note that this object is defined in the Data Model with minEntries=0 and maxEntries=unbounded. However, here the device type instead indicates support for 1 to 10 table entries for this object.

```
<object name="Device.Ethernet.Link.{i}." access="readWrite"
        minEntries="1" maxEntries="10">
  …
</object>
```

Table 64 lists the child elements allowed within an `object`. The order that these elements appear in the table is the same order, if present, that they must appear within an `object` definition.

© The Broadband Forum. All rights reserved

Table 64 – DT `object` sub-elements

Element	Multiplicity	Description
annotation	0 or 1	The object's annotation (II.2).
parameter	0 or more	Shows support for a parameter within the object (II.8).

II.8 `parameter` Element

The `parameter` element is used to specify support for a parameter defined in a DM Instance.

Possible parent elements include:

- `document/model` (II.6)
- `document/model/object` (II.7)

Table 65 lists the attributes that are available within the `parameter` element.

Table 65 – DT `parameter` attributes

Attribute	Type	Use	Description
ref	restricted xs:NCName (256)	required	The name of the parameter being referenced (maximum length 256). Each parameter name is unique within its parent `model` or `object`. Note that the parameter's object path is not specified here. **Note** – The name will have the same format as xs:NCName except that dots are not permitted; i.e. must start with a letter or "_" and subsequent characters can also include digits and connector characters such as underscore and dash. The name must in addition follow the vendor-specific object name requirements of Section 3.3/TR-106 [3].
access	One of: • readOnly • readWrite	required	Whether a parameter can be updated by the ACS.
activeNotify	One of: • normal • willDeny	optional	The parameter's active notify support. The default is *normal* if not specified.

The following example illustrates support for the PeriodicInformEnable parameter. Its `access` attribute value is the same as in the DM definition but is re-specified here since it is a required attribute. Its `activeNotify` attribute (optional) is omitted since it is unchanged from the DM definition. Sub-elements not shown here could specify annotation and syntax.

© The Broadband Forum. All rights reserved

```
<parameter ref="PeriodicInformEnable" access="readWrite">
  …
</parameter>
```

The following example illustrates support for the ParameterKey parameter. Again, its `access` attribute value is the same as in the DM definition but is re-specified here since it is a required attribute. Its `activeNotify` attribute indicates that active notification requests will be denied (while in the DM definition it is simply can deny).

```
<parameter ref="ParameterKey" access="readOnly" activeNotify="willDeny">
  …
</parameter>
```

Table 66 lists the child elements allowed within the `parameter`. The order that these elements appear in the table is the same order, if present, that they must appear within a `parameter` definition.

Table 66 – DT `parameter` sub-elements

Element	Multiplicity	Description
annotation	0 or 1	The parameter's annotation (II.2).
syntax	0 or 1	Contains the parameter's syntax supported (II.8.1). Only necessary if the supported syntax differs from what is defined in the DM definition.

II.8.1 Parameter `syntax` Element

The `syntax` element is used to specify support details for a parameter's data type. Omitting this element implies that the parameter's data type is supported exactly as it is defined in the DM Instance. The base data type is either one of the built-in data types or is derived from a named data type.

Possible parent elements include:

- `document/model/parameter` (II.8)
- `document/model/object/parameter` (II.8)

Table 67 lists the child elements allowed within a parameter's `syntax`. The `list` element if present must appear first, while the `default` element if present must appear last. One and only one of the built-in type elements, or the dataType element, can ever be present.

Table 67 – DT parameter `syntax` sub-elements

Element	Multiplicity	Description
list	0 or 1	Contains the parameter's list support (II.8.2). Only applies when the defined parameter in the DM definition is a list-valued parameter.

 © The Broadband Forum. All rights reserved

		For lists, the TR-069 parameter is always a string and the data type specification applies to individual list items, not to the parameter value.
base64 boolean dateTime hexBinary int long string unsignedInt unsignedLong	0 or 1[35]	Built-in primitive data type (II.9). Only one of these elements can be included within the `syntax`. If a built-in type is used, then the `dataType` element cannot be used. The specified type if present must be the same as in the parameter's DM definition. It is repeated here in order to indicate (using facets) how the supported data type differs from what was originally defined.
dataType	0 or 1	Reference to a named data type definition (II.8.3). If this element is used, then none of the built-in type elements can be included within the `syntax` element. The specified type if present must be the same as in the parameter's DM definition. It is repeated here in order to indicate (using facets) how the supported data type differs from what was originally defined.
default	0 or 1	Contains the parameter's default value supported (II.8.4).

Parameter `syntax` does not need to be specified unless there is something within syntax that is supported differently than is defined in the DM definition (i.e. list, data type facet, and/or default settings). See the following sections for examples on using the various `syntax` sub-elements.

II.8.2 Syntax `list` Element

The `list` element is used to specify support details for a parameter's list defined in a DM Instance. This element only applies when the defined parameter is list-valued.

Note that a list-valued parameter is always a string type, and the data type specification instead applies to individual list items not to the parameter value.

Possible parent elements include:

- `document/model/parameter/syntax` (II.8.1)
- `document/model/object/parameter/syntax` (II.8.1)

Table 68 lists the attributes that are available within the syntax's `list` element.

[35] A `syntax` element can contain either one of the built-in type elements (i.e. boolean, string, int, etc) or a `dataType` element, or neither.

 © The Broadband Forum. All rights reserved

Table 68 – DT syntax `list` attributes

Attribute	Type	Use	Description
minItems	xs:nonNegativeInteger	optional	The supported minimum number of items in the list. The default is *0* if not specified.
maxItems	xs:positiveInteger or "unbounded"	optional	The supported maximum number of items in the list. The default is *unbounded* if not specified.

Table 69 lists the child elements allowed within a syntax `list`. The `annotation` element if present must appear first.

Table 69 – DT syntax `list` sub-elements

Element	Multiplicity	Description
annotation	0 or 1	The list annotation (II.2).
size	0 or more	The supported size of the list-valued parameter, not of the individual items (II.10.1).

The following example illustrates support for a parameter syntax having a list whose items are of type unsignedInt. This list can hold between 1 and 5 items. The assumption is that the associated DM parameter will also be an unsignedInt (their base data types must match), and the list element is specified in order to restrict the number of list items supported by the device type (i.e. the DT parameter's min-max list range must fall within the range defined by the associated DM parameter's list element).

```
<syntax>
  <list minItems="1" maxItems="5"/>
  <unsignedInt/>
</syntax>
```

The following example illustrates support for a parameter syntax that is a list of MACAddress type items. This list can hold between 0 and 8 items (minItems defaults to 0 since it was omitted).

```
<syntax>
  <list maxItems="8"/>
  <dataType ref="MACAddress"/>
</syntax>
```

Note [36] – Strictly speaking, the data type elements in the above examples are not necessary since they do not alter what is already defined for the parameter in the DM definition. However, they may be included in order to provide a clearer picture of what is supported when combined with `list` and `default` elements.

[36] The `dataType/@ref` attribute will be defined in DT Schema v1.2. This will permit a parameter's data type to be referenced while still allowing it to be supported as defined in the DM parameter definition.

 © The Broadband Forum. All rights reserved

II.8.3 Syntax `dataType` Element

The syntax's `dataType` element is used to specify support details for a parameter's data type defined in a DM Instance. This element only applies when the defined parameter has a named data type (i.e. not a built-in primitive data type).

The referenced named data type must be the same as in the parameter's DM definition. It is repeated here in order to indicate (using facets) how the supported data type differs from what was originally defined.

Possible parent elements include:

- `document/model/parameter/syntax` (II.8.1)
- `document/model/object/parameter/syntax` (II.8.1)

Table 70 lists the attributes that are available within the syntax's `dataType` element.

Table 70 – DT syntax `dataType` attributes [37] [38]

Attribute	Type	Use	Description
base	restricted xs:NCName	optional	Reference to (the name of) an existing data type definition that was imported (II.4.1) into the DT Instance. This reference is made in order to define a new anonymous data type used by the parameter. The new data type can either be a restriction or extension of the named data type being referenced. Note that further content is required within this `dataType` element (i.e. one or more of the facet sub-elements will be used in order to alter the referenced definition).
ref	restricted xs:NCName	optional	Reference to (the name of) an existing data type definition that was imported (II.4.1) into the DT Instance. This will permit a parameter's data type to be referenced while still allowing it to be supported as defined in the DM parameter definition. When this attribute is specified, further content is not permitted within this `dataType` element (i.e. none of the facet sub-elements are permitted).

[37] The `dataType/@ref` attribute will be defined in DT Schema v1.2. This is analogous with the DM `syntax/dataType` element (I.10.3).

[38] A syntax `dataType` element will contain a `ref` attribute or a `base` attribute (one or the other). It is invalid for this `dataType` element to omit both the `ref` attribute and the `base` attribute.

© The Broadband Forum. All rights reserved

Note – The data type name referenced in the `base` and `ref` attributes has the same format as xs:NCName except that it cannot start with a lower-case letter (to avoid conflict with built-in data type names) and dots are not permitted; i.e. must start with an upper-case letter or "_", and subsequent characters can also include digits and connector characters such as underscore and dash.

Table 71 lists the child elements allowed within a syntax `dataType` element.

Table 71 – DT syntax `dataType` sub-elements

Element	Multiplicity	Description
size pathRef range enumeration pattern	0 or more	Data type facets (II.10).

The following example illustrates support for a parameter syntax that defines an anonymous data type based on the referenced IPAddress type (as indicated by the `base` attribute). This new data type differs from the referenced type by using the `size` facet to restrict parameter values to a maximum length of 15.

```
<syntax>
  <dataType base="IPAddress">
    <size maxLength="15"/>
  </dataType>
</syntax>
```

II.8.4 Syntax `default` Element

The `default` element indicates the default value supported for a parameter. There are two types of defaults: factory default and object default.

Possible parent elements include:

- `document/model/parameter/syntax` (II.8.1)
- `document/model/object/parameter/syntax` (II.8.1)

Table 72 lists the attributes that are available within the syntax's `default` element.

Table 72 – DT syntax `default` attributes

Attribute	Type	Use	Description
type	One of: • factory • object	required	If *factory*, the `value` attribute specifies the parameter's default value based on some standard, e.g. RFC. This applies both to static parameters as well as parameters that come about due to object creation.

 © The Broadband Forum. All rights reserved

			If *object*, the value attribute specifies the parameter's default value. This only applies to parameters that come about due to object creation.
value	xs:string	required	The value must be valid for the data type.

Table 73 lists the child elements allowed within the syntax's default element.

Table 73 – DT syntax default sub-elements

Element	Multiplicity	Description
annotation	0 or 1	The default element's annotation (II.2).

The following example illustrates support for the syntax of a boolean-valued parameter whose factory default value is *true*.

```
<syntax>
  <boolean/>
  <default type="factory" value="true"/>
</syntax>
```

The following example illustrates support for the syntax of an unsignedInt-valued parameter whose object default value is *0*. The optional annotation element provides further explanation about the default value.

```
<syntax>
  <unsignedInt/>
  <default type="object" value="0">
    <annotation>all bits clear</annotation>
  </default>
</syntax>
```

Note – Strictly speaking, the data type elements in the above examples are not necessary since they do not alter what is already defined for the parameter in the DM definition. However, they may be included in order to provide a clearer picture of what is supported when combined with list and default elements.

II.9 Built-in Primitive Data Type Elements

The DT Schema comes equipped with a set of built-in primitive data types. These are used within the syntax element to specify support details for a parameter's data type defined in a DM Instance. Use of a primitive type only applies when the defined parameter itself has a primitive type (i.e. not a named data type).

The specified primitive type must be the same as in the parameter's DM definition. It is repeated here in order to indicate (using facets) how the supported data type differs from what was originally defined for the parameter.

© The Broadband Forum. All rights reserved

Possible parent elements include:

- `document/model/parameter/syntax` (II.8.1)
- `document/model/object/parameter/syntax` (II.8.1)

For any given parent element, no more than one primitive data type sub-element is allowed (e.g. a `parameter/syntax` element can include a `boolean` sub-element or a `string` sub-element, but not both).

Note that some primitive types can be specialized using certain data type facets, as indicated in the table below.

Table 74 lists the available primitive data type elements and their permitted facets.

Table 74 – DT primitive data type elements

Element	Description
base64	Base64 encoded binary (no line-length limitation). `base64` permits zero or more of the following sub-element facets: • `size` (Length is that of the actual string, not the base64-encoded string. See Section A.2.3.3/TR-106); see II.10.1
boolean	Boolean, where the allowed values are "true" (or "1"), and "false" (or "0").
dateTime	The subset of the ISO 8601 date-time format defined by the SOAP dateTime type.
hexBinary	Hex encoded binary. `hexBinary` permits zero or more of the following sub-element facets: • `size` (Length is that of the actual string, not the hexBinary-encoded string[39]. See Section A.2.3.3/TR-106); see II.10.1
int	Integer in the range -2147483648 to +2147483647, inclusive. `int` permits zero or more of the following sub-element facets: • `range` (II.10.3)
long	Long integer in the range –9223372036854775808 to 9223372036854775807, inclusive. `long` permits zero or more of the following sub-element facets: • `range` (II.10.3)
string	Series of characters. `string` permits zero or more of the following sub-element facets: • `size` (II.10.1)

[39] In other words, it is the length of the actual string in bytes. Since a byte represents 2 hex digits, the length of the hexBinary-encoded value will be twice as many digits as the specified length.

 © The Broadband Forum. All rights reserved

	• pathRef (II.10.2) • enumeration (Each enumeration value will be unique within the string element); see II.10.4 • pattern (Each pattern value will be unique within the string element); see II.10.5
unsignedInt	Unsigned integer in the range 0 to 4294967295, inclusive. unsignedInt permits zero or more of the following sub-element facets: • range (II.10.3)
unsignedLong	Unsigned long integer in the range 0 to 18446744073709551615, inclusive. unsignedLong permits zero or more of the following sub-element facets: • range (II.10.3)

The following example lists all primitive types and their sub-element facets. Note that this is not a practical example, since these elements would actually appear independently within parameter syntax.

```
<base64>
  <size …/>
</base64>

<boolean/>

<dateTime/>

<hexBinary>
  <size …/>
</hexBinary>

<int>
  <range …/>
</int>

<long>
  <range …/>
</long>

<string>
  <size …/>
  <pathRef …/>
  <enumeration …/>
  <pattern …/>
</string>

<unsignedInt>
  <range …/>
</unsignedInt>

<unsignedLong>
  <range …/>
</unsignedLong>
```

 © The Broadband Forum. All rights reserved

II.10 Data Type Facets

Facet elements can be used within a parameter's data type syntax to specify different aspects of how the data type is to be supported, such as size, range, etc.

Possible parent elements include:

- `document/model/parameter/syntax/dataType` (II.8.3)
- `document/model/object/parameter/syntax/dataType` (II.8.3)
- built-in primitive data types (II.9)

Note that some of the built-in primitive data types support a limited use of facets. This is discussed in Section II.9.

Table 75 lists the available facet elements. See Section A.2.3.3/TR-106 [3] for details concerning the definition of DM-based facet types (which may be informative, but note that they do differ slightly for DT facets as is shown in the sections below).

Note – For any given parent element, zero or more of each facet sub-element is permitted unless otherwise indicated. The convention is to include only those facet elements that indicate a change in support as compared to the parameter definition in the DM Instance.

Table 75 – DT data type facet elements

Element	Description
size	The minimum and maximum length for a string-related type (II.10.1). Multiple such elements can be used to indicate length ranges.
pathRef	Reference to an object or parameter via its path name string (II.10.2).
range	The minimum and maximum value for an integer-related type (II.10.3). Multiple such elements can be used to define disjoint integer ranges.
enumeration	Specific value that is valid within a string-related type (II.10.4). Multiple such elements can be used to define a set of valid values.
pattern	Pattern of valid values for a string-related type (II.10.5). Multiple such elements can be used to specify different patterns within a type definition.

Table 76 lists the child elements allowed within all of the facet elements.

Table 76 – DT facet sub-elements

Element	Multiplicity	Description
annotation	0 or 1	The facet's annotation (II.2).

II.10.1 `size` Element

Size facets, taken together, define the valid size ranges (i.e. string lengths), e.g. (0:0) and (6:6) mean that the size has to be 0 or 6. The `size` facet can only be specified for string data types, i.e. data types that are derived from base64, hexBinary or string.

 © The Broadband Forum. All rights reserved

Table 77 lists the attributes that are available within a `size` element. Note that this is a subset of the attributes defined by a DM size facet (I.13.1).

Table 77 – DT `size` attributes

Attribute	Type	Use	Description
access	One of: • readOnly • readWrite	optional	The default is *readWrite* if not specified.
minLength	xs:nonNegativeInteger	optional	The default is *0* if not specified.
maxLength	xs:nonNegativeInteger	optional	The default is *16* when not specified and no implied maximum exists (see Section 3.2.6/TR-106 [3] for further details).

Note – The `access` attribute serves no purpose. It should not be used and may be removed in a future version of the DT Schema.

The following example illustrates support for a string length size between 1 and 255 characters.

```
<size minLength="1" maxLength="255"/>
```

II.10.2 `pathRef` Element

PathRef facets specify how a parameter can reference another parameter or object via a path name. The `pathRef` facet can only be specified for data types that are derived from string (i.e. string and its derived types).

Table 78 lists the attributes that are available within a `pathRef` element. Note that this is a subset of the attributes defined by a DM pathRef facet (I.13.3).

Table 78 – DT `pathRef` attributes

Attribute	Type	Use	Description
targetParent	list of Path Names	optional	An XML list of path names that can restrict the set of parameters and objects that can be referenced. The list cannot be empty. 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, and can therefore reference multiple objects. Path names cannot contain explicit instance numbers. Each path name will follow the

© The Broadband Forum. All rights reserved

			requirements of Section A.2.3.4/TR-106 (with path name scope specified by the `targetParentScope` attribute in the associated DM Instance facet definition[40]).
targetType	One of: • any • parameter • object • single • table • row	optional	Specifies the type of item that can be referenced by `targetParent` (see Section A.2.3.7/TR-106). **any**: Either a parameter or an object can be referenced. **parameter**: Only a parameter can be referenced. **object**: Any type of object can be referenced. **single**: Only a single-instance object can be referenced. **table**: Only a multi-instance object (table) can be referenced. **row**: Only a multi-instance object instance (table row) can be referenced. The default is *any* if not specified.
targetDataType	One of: • any • base64 • boolean • dateTime • hexBinary • integer • int • long • string • unsignedInt • unsignedLong • <named data type>	optional	Specifies the valid data types for a referenced parameter (see Section A.2.3.7/TR-106). Is relevant only when `targetType` is *any* or *parameter*; if the DT targetType is omitted, then the DM targetType is considered. The default is *any* if not specified. For named data types, see Section II.4.1. For primitive data types, see Section II.9. 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

[40] In the absence of a DM targetParentScope, the targetParent path name itself can imply scope (see I.13.3 Table 45).

 © The Broadband Forum. All rights reserved

			numeric data type".

The following example illustrates support for a `pathRef` which can only reference a boolean-typed parameter.

```
<pathRef targetType="parameter" targetDataType="boolean"/>
```

The following example illustrates support for a `pathRef` which can only reference an object instance (row) within the Bridge or VLAN tables. Note that the `targetParent`'s path names start with a dot and therefore imply that they are relative to the Root (or Service) Object.

```
<pathRef targetParent=".Bridging.Bridge. .Bridging.Bridge.{i}.VLAN."
         targetType="row"/>
```

The following example illustrates support for a `pathRef` which can only reference a row within the Profile table. Since the `targetParent` attribute value does not start with "Device" or "InternetGatewayDevice" or a dot, the target path name is relative to the current object by default.

```
<pathRef targetParent="Profile." targetType="row"/>
```

II.10.3 `range` Element

Range facets, taken together, define the valid value ranges, e.g. [-1:-1] and [1:4094] mean that the value has to be -1 or 1:4094 (it cannot be 0). The `range` facet can only be specified for numeric data types, i.e. data types that are derived from one of the integer types.

Table 79 lists the attributes that are available within a `range` element. Note that this is a subset of the attributes defined by a DM range facet (I.13.4).

Table 79 – DT `range` attributes

Attribute	Type	Use	Description
access	One of: • readOnly • readWrite	optional	Whether values within the specified range can be can be written by the ACS. The default is *readWrite* if not specified.
minInclusive	xs:integer	optional	Minimum value in the range. If omitted, the default minimum is the minimum allowed by the base type.
maxInclusive	xs:integer	optional	Maximum value in the range. If omitted, the default maximum is the maximum allowed by the base type.

 © The Broadband Forum. All rights reserved

The following example illustrates support for a range where the parameter value can be -1 or between 1 and 10. Zero is not a valid value. Also, the ACS can only write values between 1 and 10.

```
<range access="readOnly" minInclusive="-1" maxInclusive="-1">
<range minInclusive="1" maxInclusive="10">
```

II.10.4 `enumeration` Element

Enumeration facets, taken together, define the valid values, e.g. "a" and "b" mean that the value has to be a or b. The `enumeration` facet can only be specified for data types that are derived from string. Derived types may add additional enumeration values. As a reference point, see Section A.2.5/TR-106 for how this is done in DM Instances.

Table 80 lists the attributes that are available within an `enumeration` element. Note that this is a subset of the attributes defined by a DM enumeration facet (I.13.5).

Table 80 – DT `enumeration` attributes

Attribute	Type	Use	Description
access	One of: • readOnly • readWrite	optional	Whether an enumeration value can be written by the ACS. The default is *readWrite* if not specified.
value	xs:string	required	An enumeration value. Duplicate values are not allowed within the associated parameter.

The following example illustrates support for the set of enumeration values: None, Requested, Complete, and Error. Each enumeration value is unique to the set. Only the Requested value can be written by the ACS to the associated parameter.

```
<enumeration value="None" access="readOnly"/>
<enumeration value="Requested"/>
<enumeration value="Complete" access="readOnly"/>
<enumeration value="Error" access="readOnly"/>
```

II.10.5 `pattern` Element

Pattern attributes, taken together, define valid patterns, e.g. "" and "[0-9A-Fa-f]{6}" means that the value has to be empty or a 6 digit hex string. The `pattern` facet can only be specified for data types that are derived from string.

Note – The pattern syntax is the same as for XML Schema regular expressions. See XML Schema Part 2 [12] Appendix F.

Table 81 lists the attributes that are available within a `pattern` element. Note that this is a subset of the attributes defined by a DM pattern facet (I.13.7).

© The Broadband Forum. All rights reserved

Table 81 – DT `pattern` attributes

Attribute	Type	Use	Description
access	One of: • readOnly • readWrite	optional	Whether a parameter value that matches the pattern value can be written by the ACS. The default is *readWrite* if not specified.
value	xs:string	required	Pattern for the data type (a regular expression).

The following example illustrates support for a set of five patterns for some data type or parameter. The first four patterns are constant values. The last pattern defines a value which includes an "X", a space, any six digit hex number, a space, followed by any sequence of characters.

```
<pattern value="1 Firmware Upgrade Image">
<pattern value="2 Web Content">
<pattern value="3 Vendor Configuration File">
<pattern value="4 Vendor Log File">
<pattern value="X [0-9A-F]{6} .*">
```

 © The Broadband Forum. All rights reserved

Appendix III – Reference: Device Type Features XML Schema

This appendix provides a user reference for the TR-069 device type features schema (DTF Schema). The normative version can be found at http://www.broadband-forum.org/cwmp/cwmp-devicetype-features.xsd.

The DTF Schema consists of a list of feature names that can be used within DT Instance documents, specifically, within their `feature` elements (II.3). Each feature specified by a DT Instance document indicates support for that feature.

Note that DT Instance documents do not need to declare their intent to use DTF feature names. This is because the DT Schema itself references the DTF Schema in order to incorporate these feature names directly (i.e. to a DT Instance document, the feature names appear to be part of the DT Schema).

III.1 Feature Names

The DTF Schema simply defines a list of feature names, which are referenced from the DT Schema's `feature` element (II.3). In this way, the list of possible feature names can be updated (within the DTF Schema) independent of the DT Schema.

The list of feature names is:

- DNSClient
- DNSServer
- Firewall
- IPv6
- NAT
- Router
- X_<VENDOR>_VenderSpecificName

The last item in the above list indicates that the DTF Schema allows vendors to define their own feature names. Vendor-specific feature names will begin with X_<VENDOR>_, where <VENDOR> must be as defined in Section 3.3/TR-106 [3].

The following example snippet illustrates the use of feature names that might appear within a DT Instance document. Each feature element indicates support for a different named feature.

```
<feature name="Router"/>
<feature name="IPv6"/>
<feature name="X_ACDC73_VPN"/>
```


© The Broadband Forum. All rights reserved

Appendix IV – Reference: Data Model Report XML Schema

This appendix provides a user reference for the TR-069 Data Model report schema (DMR Schema). The official version (non-normative) can be found at http://www.broadband-forum.org/cwmp/cwmp-datamodel-report.xsd.

The DMR Schema consists of a set of attributes that can be used within DM or DT Instance documents to provide additional information to reporting tools. However, there is no direct dependency between the DMR Schema and the DM or DT Schemas. Rather, it is up to DM and DT Instances to declare their intent to use DMR attributes.

Note that the DMR namespace is *urn:broadband-forum-org:cwmp:datamodel-report-0-1*. The fact that its namespace includes the version number "*0-1*" is an unfortunate artifact; updates to the DMR Schema will not carry a revision number (i.e. the "*0-1*" is included in the namespace for backward compatibility only and will not be incremented when the schema is revised).

IV.1 DMR Attributes

The following attributes can be used within DM and DT Instances to provide additional information to reporting tools. The DMR Schema does not formally stipulate which elements these attributes can be used with (reporting tools have some latitude), though, some guidance can be inferred from the attribute descriptions and how these attributes are being used today within the published CWMP XML Data Models.

Note that these are all optional attributes.

Table 82 – DMR attributes

Attribute	Type	Description
fixedObject	xs:boolean	This attribute is meant to be used with an `object` element. It indicates that the object is fixed, and so a report tool should not (for example) warn about any writable parameters that the object might contain.
hideDeleted	xs:boolean	This attribute is meant to be used with elements that support the `status` attribute (e.g. `object`, `parameter`, `enumeration`) and have their status set to "deleted". It indicates that a deleted item should always be hidden in the report, e.g. not even shown in strikeout.
noUniqueKeys	xs:boolean	This attribute is meant to be used with an `object` element whose `maxEntries` attribute is greater than 1 (i.e. a table).

 © The Broadband Forum. All rights reserved

		It indicates that the object has no unique keys (and is not meant to have unique keys), so a report tool should not (for example) warn about the lack of unique keys.
previousParameter	restricted xs:NCName (256)	This attribute is meant to be used with a `parameter` element. Previous parameter hint. It indicates the name of the existing parameter that should come immediately before the parameter adorned by this attribute. This is useful when the desired previous parameter is defined in a different file from the parameter in question. **Note** – The attribute value has the same format as xs:NCName except that dots are not permitted; i.e. must start with a letter or "_" and subsequent characters can also include digits and connector characters such as underscore and dash.
previousObject	restricted xs:NCName (256)	This attribute is meant to be used with an `object` element. It also makes sense to use it with a referencing `component` element (i.e. `model/component`) if the component contains an object. Previous object hint. It indicates the name of the existing object that should come immediately before the object adorned by this attribute. This is useful when the desired previous object is defined in a different file from the object in question. **Note** – Each dot-separated portion of the (overall object name specified in the) attribute value has the same format as xs:NCName except that dots are not permitted; i.e. must start with a letter or "_" and subsequent characters can also include digits and connector characters such as underscore and dash.
previousProfile	restricted xs:NCName	This attribute is meant to be used with a `profile` element. It also makes sense to use it with a referencing `component` element (i.e. `model/component`) if the component contains a profile. Previous profile hint. It indicates the name of the existing profile that should come immediately before the profile adorned by this attribute. This is useful when the desired previous profile is defined in a different file from the profile in question. **Note** – The attribute value is formatted as

© The Broadband Forum. All rights reserved

		"<name>:<version>". The <name> part is the same as xs:NCName except that dots are not permitted; i.e. must start with a letter or "_" and subsequent characters can also include digits and connector characters such as underscore and dash. The <version> part contains digits only.
version	restricted xs:token	It indicates the desired version number of the item in question, consisting of a major and minor version. This attribute can be used with any element for which versioning makes sense to reporting tools. However, many elements already have an intrinsic concept of version, so care should be taken. For example, `model` and `profile` elements specify a version as part of their name; while `object` and `profile` elements already inherit a version from their parent `model` element. **Note** – The attribute value is formatted as "<major-version>.<minor-version>", where the <major-version> and <minor-version> parts are digits. **Note** – The only known use for the `version` attribute is in the "flattened" XML, where the original version info is no longer available.

The following example illustrates the use of the `previousParameter` attribute within a DM Instance document. Its value provides a hint to the Report Tool that the IPv4Enable parameter should follow the (previously defined) Enable parameter within the IP.Interface object.

```
<object base="Device.IP.Interface.{i}." ...>
  <parameter name="IPv4Enable" access="readWrite"
             dmr:previousParameter="Enable">
    ...
</object>
```

In this example, `previousParameter` has an empty value which hints to the Report Tool that the EnableCWMP parameter should appear in the report as the first parameter under the ManagementServer object.

```
<component ...>
  <object base="ManagementServer." ...>
    <parameter name="EnableCWMP" access="readWrite" dmr:previousParameter="">
    ...
  </object>
</component>
```

In this example, `previousProfile` hints that the IPInterface:2 profile should follow the existing IPInterface:1 profile in the report. And under the IPInterface:2 profile,

© The Broadband Forum. All rights reserved

`previousParameter` hints that the IPv4Enable parameter should follow the existing IPv4Capable parameter within the IP object.

```
<profile name="IPInterface:2" base="IPInterface:1"
         dmr:previousProfile="IPInterface:1">
  <object ref="Device.IP." ...>
    <parameter ref="IPv4Enable" requirement="readWrite"
               dmr:previousParameter="IPv4Capable"/>
    ...
  </object>
</profile>
```

In this example, `previousObject` hints that the DHCPv6 object should follow the existing DHCPv4 object in the report.

```
<object name="Device.DHCPv6." access="readOnly" minEntries="1" maxEntries="1"
        dmr:previousObject="Device.DHCPv4.">
  ...
</object>
```

In this example, `previousObject` hints that the objects defined within the referenced UDPEchoConfig component should follow the existing UploadDiagnostics object in the report. Similarly, `previousProfile` hints that the profiles defined within the referenced UDPEchoConfig component should follow the existing UploadTCP:1 profile.

```
<component path="Device.IP.Diagnostics." ref="UDPEchoConfig"
           dmr:previousObject="UploadDiagnostics."
           dmr:previousProfile="UploadTCP:1"/>
```

In this example, `noUniqueKeys` hints that the Report Tool should not warn that there are no unique keys defined for DeviceInfo.Processor object table. This is simply a matter of suppressing the warnings when the report is being generated.

```
<component ...>
  <object name="DeviceInfo.Processor.{i}." access="readOnly"
          minEntries="0" maxEntries="unbounded"
          numEntriesParameter="ProcessorNumberOfEntries"
          dmr:noUniqueKeys="true">
    ...
  </object>
</component>
```

In this example, `fixedObject` hints that the DSL.Line table is fixed. This is useful here because a DSL line models a physical interface whose instances should always be present in the associated CPE.

```
<object name="Device.DSL.Line.{i}." access="readOnly"
        numEntriesParameter="LineNumberOfEntries"
        minEntries="0" maxEntries="unbounded"
        dmr:fixedObject="true">
```

 © The Broadband Forum. All rights reserved

```
  ...
</object
```

 © The Broadband Forum. All rights reserved

Appendix V – Processing Data Models, Validating and Reporting

This appendix briefly discusses XML tools, validating Data Models, and generating the HTML reports that are published alongside an XML Data Model.

V.1 XML Schemas and Data Model Definitions

All of the published Broadband Forum Data Models and schema files can be downloaded from http://www.broadband-forum.org/cwmp.zip.

V.2 XML Editor

Using an XML editor is optional but can be helpful. While any text editor can be used for editing XML, it is worth obtaining an editor that understands XML syntax and XML Schema. Such an editor can support context-sensitive editing and will probably have a built-in XML Schema verifier.

The following are some notable XML editors. These are commercial editors; their inclusion here should not be taken as an endorsement:

- Altova XMLSpy
  http://www.altova.com/products/xmlspy/xml_editor.html (Windows)
- Oxygen XML Editor
  http://www.oxygenxml.com (Windows, Mac OS X, Linux)

See http://en.wikipedia.org/wiki/Category:XML_editors for further suggestions.

V.3 XML Schema Verifier

Both XMLSpy and Oxygen include XML Schema verifiers. However, it can be useful to have access to standalone verifiers, e.g. for automated verification in makefiles. Note that the Report Tool, discussed in Section V.4, can also be run with schema validation enabled.

The following are some standalone XML Schema verifiers. These are free but licensed tools, available on a variety of operating systems:

- xmllint (part of libxml2)
  http://xmlsoft.org
- xsv (W3C Validator) -
  http://www.w3.org/2001/03/webdata/xsv (online)
  http://www.ltg.ed.ac.uk/~ht/xsv-status.html (tool)

For example, this is how xmllint can be run from the command-line to verify tr-106-1-0-0.xml:

```
% xmllint --noout --schema cwmp-datamodel-1-0.xsd tr-106-1-0-0.xml

tr-106-1-0-0.xml validates
```


© The Broadband Forum. All rights reserved

However, this will not work with some of the newer XML Data Models because they are associated with more than one schema (e.g. tr-181-1-0-0.xml is associated with a DM Schema and a DMR Schema). This can be overcome by defining a schema file that combines DM and DMR Schemas into one XSD file. The following listing illustrates such a cwmp-all-schemas.xsd file.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:tns="urn:broadband-forum-org:cwmp:all"
           targetNamespace="urn:broadband-forum-org:cwmp:all"
           elementFormDefault="unqualified"
           attributeFormDefault="unqualified">
  <xs:import namespace="urn:broadband-forum-org:cwmp:datamodel-1-0"
             schemaLocation="cwmp-datamodel-1-0.xsd"/>
  <xs:import namespace="urn:broadband-forum-org:cwmp:datamodel-1-1"
             schemaLocation="cwmp-datamodel-1-1.xsd"/>
  <xs:import namespace="urn:broadband-forum-org:cwmp:datamodel-1-2"
             schemaLocation="cwmp-datamodel-1-2.xsd"/>
  <xs:import namespace="urn:broadband-forum-org:cwmp:datamodel-1-3"
             schemaLocation="cwmp-datamodel-1-3.xsd"/>
  <xs:import namespace="urn:broadband-forum-org:cwmp:datamodel-1-4"
             schemaLocation="cwmp-datamodel-1-4.xsd"/>
  <xs:import namespace="urn:broadband-forum-org:cwmp:datamodel-report-0-1"
             schemaLocation="cwmp-datamodel-report.xsd"/>
</xs:schema>
```

Now xmllint can verify tr-181-1-0-0.xml using cwmp-all-schemas.xsd:

```
% xmllint --noout --schema cwmp-all-schemas.xsd tr-181-1-0-0.xml

tr-181-1-0-0.xml validates
```

V.4 BBF Report Tool

The BBF Report Tool is a Perl script that can be used to scan a TR-069 XML Data Model for errors and generate its reports. Various formats can be generated, but it is most common to generate HTML Data Model reports.

HTML reports are non-normative, but are published on the Broadband Forum website alongside their normative XML in order to provide a human readable view. Note that the Report Tool can also perform schema validation (writing errors to standard error and to the HTML output).

The Report Tool can be downloaded from the UNH XML tools repository, which lives at https://tr69xmltool.iol.unh.edu/repos/cwmp-xml-tools/Report_Tool/.

V.4.1 Command Line Version

The easiest way to run the Report Tool from the command line is to use its standalone binary (i.e. report.exe for Windows, or report.darwin for Mac). No need to have Perl installed.

© The Broadband Forum. All rights reserved

Alternatively, the Report Tool script (report.pl) can be run with Perl directly, which requires some additional modules to be installed. In a Unix-like environment you can use the standard `cpan` command to do this as follows:

```
% cpan Algorithm::Diff Clone Config::IniFiles File::Compare URI::Escape XML::LibXML
```

If you want to use `cpan` under Windows to retrieve the additional modules, Strawberry Perl (http://strawberryperl.com) is the recommended free Perl engine. After installing Strawberry Perl on Windows, simply perform the `cpan` command listed above.

The Report Tool supports many options, e.g. to control the report format, to highlight the differences between two files, or to request more pedantic warnings. The list of options is described at https://tr69xmltool.iol.unh.edu/repos/cwmp-xml-tools/Report_Tool/README.txt. Alternatively, you can run the Report Tool with the --help option to output an explanation of all of the options:

```
% report.pl --help
```

Only a few of these options are needed to generate the HTML reports that are published. The most common options are --include, --pedantic, and --report=html. For example, the following commands could be used to generate the HTML reports for TR-181 Issue 2 Amendment 1 (i.e. tr-181-2-1-0.html and tr-181-2-1-0-last.html).

```
% report.exe --report=html --include=cwmp --showreadonly --warnbibref
--pedantic tr-181-2-1-0.xml > tr-181-2-1-0.html
```

```
% report.exe --report=html --include=cwmp --showreadonly --warnbibref
--pedantic --lastonly tr-181-2-1-0.xml > tr-181-2-1-0-last.html
```

The options that are used above are as follows:

- --report=html : generates an HTML report (to standard output) that is intended to be as similar as possible to the old Word tables, but with the added benefits of a table of contents and hyperlinks. By default, i.e. if the --report option is omitted, no report is generated, and the Report Tool just reads the XML files and checks for errors. Other report types include tab, text, xls, xml, and xsd.
- --include=d… : can be specified multiple times; specifies directories to search for files specified on the command line or included from other files; the current directory is always searched first. No search is performed for files that already include directory names.
  Note: in the example above, the published Broadband Forum XML/XSD files (see V.1) have been placed in the “cwmp” directory. If all required files were placed together in the current directory, this option would not be necessary.
- --showreadonly : shows read-only enumeration and pattern values as READONLY.
- --warnbibref : enables bibliographic reference warnings. Can give the option a value to indicate the level of warnings; the higher the level the more warnings (--warnbibref is the same as --warnbibref=1).

 © The Broadband Forum. All rights reserved

- --pedantic : outputs warnings (to standard error) when logical inconsistencies in the XML are detected. This option also enables XML Schema validation of DM Instances. Can give the option a value to indicate the level of pedantic; the higher the level the more pedantic (--pedantic is the same as --pedantic=1).
- --lastonly : reports only on items that were defined or modified in the last (in this case only) XML file on the command line. This is useful when you want only to see the changes that the last file made.

During the development of a draft Data Model, it can be helpful to create a report that indicates what changed since the previous version of the Data Model. The following example generates an HTML report that includes change marks, highlighting changes between draft tr-181-2-4-0.xml and tr-181-2-3-0.xml (the previous version of the Data Model).

```
% report.exe --report=html --include=cwmp --showreadonly --warnbibref
--pedantic --lastonly --showdiffs tr-181-2-4-0.xml > tr-181-2-3-0-last.html
```

The additional options that are used above are as follows:

- --showdiffs : affects only the text and html reports; visually indicates the differences resulting from the last XML file on the command line. For the html report, insertions are shown in blue and deletions are shown in red strikeout.

Note that when the Report Tool runs, it will output messages indicating its progress. These include errors, warnings, and general information. Error lines are prefixed with an (E), warning lines are prefixed with a (W), and informational lines are prefixed with an (I).

```
% report.pl --report=xml --include=cwmp tr-181-2-2-0.xml > tr-181-2-2-0-full.xml

(I) urn:broadband-forum-org:tr-143-1-0-2: 41
(I) urn:broadband-forum-org:tr-157-1-0-0: 230
(I) urn:broadband-forum-org:tr-157-1-1-0: 5
(I) urn:broadband-forum-org:tr-157-1-2-0: 9
(I) urn:broadband-forum-org:tr-157-1-3-0: 76
(I) urn:broadband-forum-org:tr-181-2-0-1: 1718
(I) urn:broadband-forum-org:tr-181-2-1-0: 1
```

In the above example, the URNs and numbers (in the information output lines) indicate where the various objects and parameters were originally defined. Also note that this example is using the `report=xml` report type; this will produce a consolidated XML report (i.e. generate a single XML file that combines tr-181-2-2-0.xml, with all of its supporting XML files, into one document).

V.4.2 GUI Version

A graphical frontend to the Report Tool is also available for Windows[41]. An executable installer (ReportGuiSetup.exe) can be downloaded from https://tr69xmltool.iol.unh.edu/repos/cwmp-xml-tools/ReportGui/. The installer includes the report.exe and a client GUI.

[41] XP, Vista, and Windows 7

 © The Broadband Forum. All rights reserved

The GUI has a "standard report function" tab (where you simply select the type of report you want without the confusion of selecting specific report options) and an "expert report functions" tab (where you can select from a canned set of report options, or even manually type in any report option).

Creating a report is as simple as selecting which XML Data Model file to use, selecting the desired options (either in the standard or expert tab), and then selecting the generate button.

Additional information about the report GUI and its options can be found online at https://tr69xmltool.iol.unh.edu/wiki/ReportGUI.

V.5 Published BBF Data Model Reports

Each DM Data Model will have the following reports published on the Broadband Forum website alongside their normative XML:

- **HTML Full** – Represents an entire Data Model (e.g. if reporting on the Device:2.3 XML, the HTML report file will include this and all previous revisions of the Device:2 Data Model).

  To generate this report, the Report Tool must be run with the following options:
  `--report=html`

- **HTML Diff** – Represents the latest revision of a Data Model (e.g. if reporting on the Device:2.3 XML, the HTML report file will only include the changes defined in that revision and will exclude unrelated definitions from all previous revisions of the Device:2 Data Model).

  To generate this report, the Report Tool must be run with the following options:
  `--report=html --lastonly`

- **XML Full** – Aggregates the group of XML files that together define a given Data Model, and flattens them into one XML file. This XML format is easier for vendors to work with.

  To generate this report, the Report Tool must be run with the following options:
  `--report=xml`

See Section V.4 for additional information on generating reports with the Report Tool.

© The Broadband Forum. All rights reserved

Index

A

annotation, DT, 139, 197
Anonymous Data Type, 83, 85, 172, 173, 180, 209, 210

B

Base Type Restriction, 60, 85, 102
Baseline Profiles, 33, 110, 111
bibliography, 54, 155
  Citing, 56
  reference, 55, 56, 156
bibliography, DT, 123, 200
  reference, 201

C

Central Bibliography File, 21, 54, 56
Central Data Types File, 21, 58, 62, 124
Comma-Separated List. *See* parameter / list
component, 103, 157
  Importing, 106
  Referencing (including), 104, 106, 107, 158
  Updating, 106, 108

D

Data Model, 16, 19, 21, 54, 122, 226
  Updates, 36, 47, 67, 78, 80, 100, 102, 111
  Versioning, 20
Data Type Facets, 84, 134, 182, 214
dataType, 58, 59, 154
description, 114, 142
  Character Set, 114
  Updating, 116
  Whitespace, 117, 143
DM Schema, 16, 24, 141
document, 24, 141
  file, 25, 141
  spec, 24, 67, 68, 141
document, DT, 196
DT Schema, 16, 196

E

Enable Parameter, 30, 75

F

feature, DT, 140, 197, 220

I

import, 61, 151
  bibliography, 32, 61, 62
  component, 63, 152
    underscore prefix, 65, 107
  dataType, 62, 152
  model, 62, 152
import, DT, 124, 198
  bibliography, 124
  dataType, 124, 199
  model, 125, 199

M

Markup, 118, 139, 144
model, 66, 159
  component, 105
  Errata, 68
  Updating, 67
  Versioning, 66, 68
model, DT, 127, 202

N

Named Data Types, 83, 171, *See* dataType
Normal Path Name Scope, 94, 95, 97, 99
NumberOfEntries Parameter, 29, 30, 73, 76, 89

O

object, 70, 161
  enableParameter, 30, 74, 75, 76, 78, 163
  Hierarchy, 70
  Multi Instance, 72
    Fixed Sized, 76
    Variable Sized, 73, 74, 89
  numEntriesParameter, 29, 30, 73, 76, 89, 163
  Single Instance, 70
    1 of n, 71
  uniqueKey, 76, 165
  Updating, 78
object, DT, 129, 203
  Multi Instance, 129
  Single Instance, 129

P

parameter, 81, 166
  activeNotify, 88
  command, 90
  dataType, 83, 171
  default, 87, 173

 © The Broadband Forum. All rights reserved

forcedInform, 88
hidden, 89
list, 91, 170
enumeration, 92
Fixed Size, 93
Variable Size, 91, 92
syntax, 168
Updating, 100
parameter, DT, 132, 205
dataType, 209
default, 135, 210
list, 136, 207
pathRef, 137
syntax, 133, 206
Path Name, 16, 70, 71, 72, 94
Primitive Data Types, 82, 180, 211
profile, 110, 174
Extending, 112
object, 177
parameter, 178
Updating, 111

R

Reference Parameter, 93
enumerationRef, 93, 98, 192
instanceRef, 93, 97, 184
pathRef, 93, 94, 97, 186
pathRef list, 96
Relative Path Name, 94, 95, 96, 98, 99
Report Tool, 16, 55, 56, 69, 117, 118, 143, 144, 145, 221, 227
GUI, 229
Reports, 228, 230
Root Data Model. *See* model
Defining, 27
Updating, 35

S

Service Data Model. *See* model
Defining, 41
Updating, 46
status (deprecate, obsolete, delete), 80, 102, 112, 120
Supported Data Model, 122, 127, 128

T

Templates, 17, 28, 118, 139, 145

U

User Guide Conventions, 13

V

Vendor-Specific Data Model, 52, 53, 54, 125

X

XML
Attribute, 13
Catalog, 24, 25
Element, 13
File Names, 19, 25, 141
Published Files, 21
Root Element, 24, 141, 196

End of Broadband Forum Technical Report TR-154

© The Broadband Forum. All rights reserved