TECHNICAL REPORT

TR-181
Device Data Model for TR-069

Issue: 2
Issue Date: May 2010

Notice

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

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

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

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

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

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

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

Issue History

Issue Number	Issue Date	Issue Editor	Changes
Issue 2	May 2010	Paul Sigurdson, Broadband Forum William Lupton, 2Wire	Original. Defines version 2 of the TR-069 Device data model (Device:2).

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

Editors:	William Lupton	2Wire
	Paul Sigurdson	Broadband Forum
BroadbandHome™ Working Group Chairs	Greg Bathrick	PMC-Sierra
	Heather Kirksey	Alcatel-Lucent

Table of Contents

List of Figures

List of Tables

Executive Summary

This Technical Report defines version 2 of the TR-069 [2] Device data model (Device:2). The Device:2 data model applies to all types of TR-069-enabled devices, including End Devices, Internet Gateway Devices, and other Network Infrastructure Devices. It represents a next generation evolution that supersedes both Device:1 and InternetGatewayDevice:1.

The evolution to Device:2 was necessary in order resolve some fundamental limitations in the InternetGatewayDevice:1 data model, which proved to be inflexible and caused problems in representing complex device configurations. However, in defining this next generation data model, care has been taken to ensure that all InternetGatewayDevice:1 and Device:1 functionality has been covered. Legacy installations can continue to make use of the InternetGatewayDevice:1 and Device:1 data models, which are still valid.

The Device:2 data model defined in this Technical Report consists of a set of data objects covering things like basic device information, time-of-day configuration, network interface and protocol stack configuration, routing and bridging management, throughput statistics, and diagnostic tests. It also defines a baseline profile that specifies a minimum level of data model support.

The cornerstone of the Device:2 data model is the interface stacking mechanism. Network interfaces and protocol layers are modeled as independent data objects that can be stacked, one on top of the other, into whatever configuration a device might support.

1 Purpose and Scope

1.1 Purpose

This Technical Report defines version 2 of the TR-069 [2] Device data model (Device:2). The Device:2 data model applies to all types of TR-069-enabled devices, including End Devices, Internet Gateway Devices, and other Network Infrastructure Devices. It represents a next generation evolution that supersedes both Device:1 and InternetGatewayDevice:1.

The evolution to Device:2 was necessary in order resolve some fundamental limitations in the InternetGatewayDevice:1 data model, which proved to be inflexible and caused problems in representing complex device configurations. However, in defining this next generation data model, care has been taken to ensure that all InternetGatewayDevice:1 and Device:1 functionality has been covered. Legacy installations can continue to make use of the InternetGatewayDevice:1 and Device:1 data models, which are still valid.

1.2 Scope

The Device:2 data model defined in this Technical Report consists of a set of data objects covering things like basic device information, time-of-day configuration, network interface and protocol stack configuration, routing and bridging management, throughput statistics, and diagnostic tests. It also defines a baseline profile that specifies a minimum level of data model support.

The cornerstone of the Device:2 data model is the interface stacking mechanism. Network interfaces and protocol layers are modeled as independent data objects (a.k.a. interface objects) that can be stacked, one on top of the other, into whatever configuration a device might support.

Figure 1 illustrates the top-level Device:2 data model structure. Figure 2, Figure 3, and Figure 4 illustrate the data model structure in greater detail.

- Interface objects are indicated by a “dashed” background pattern.
- Objects that reference interface objects are indicated by a “dotted” background pattern.

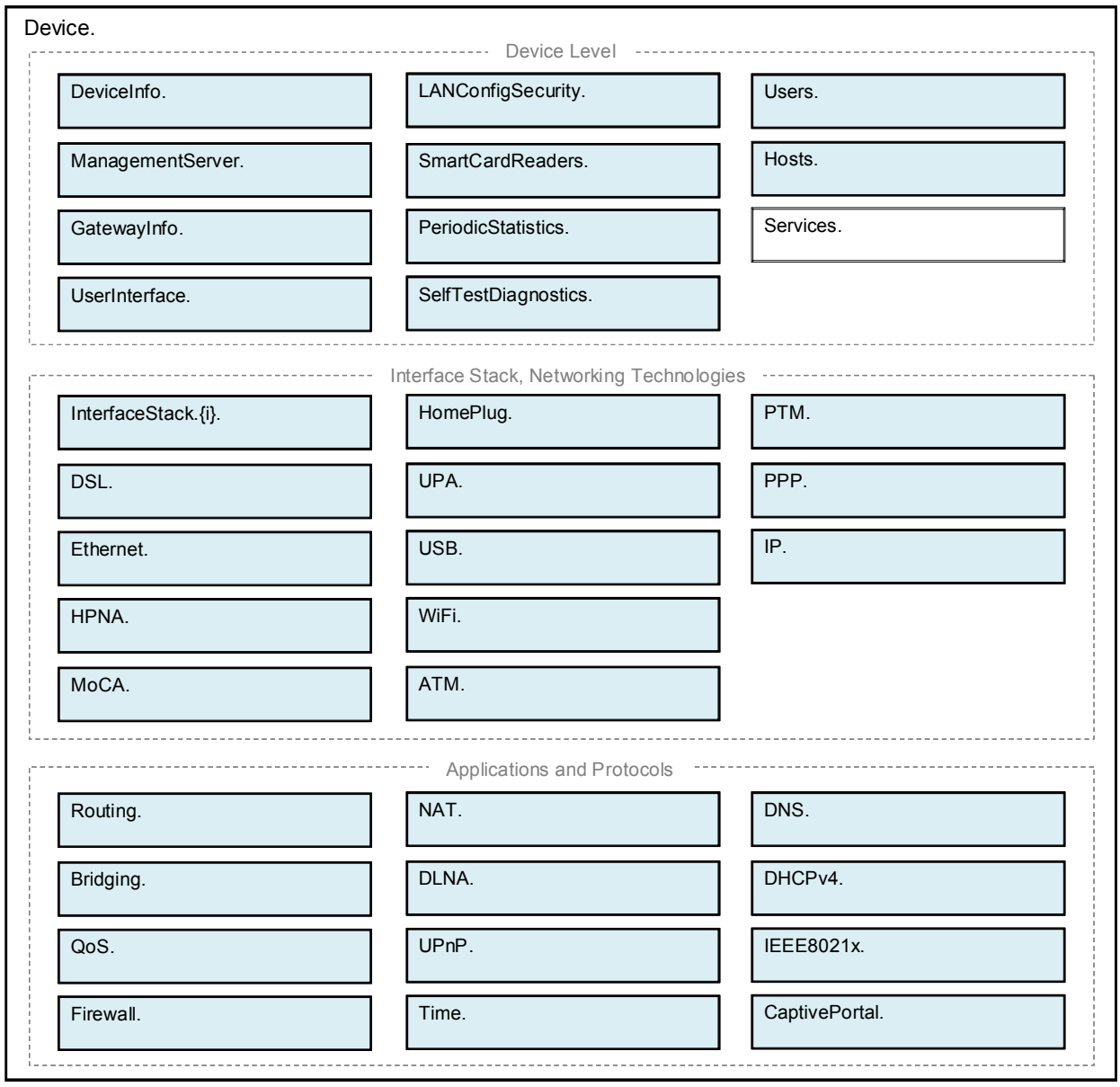

Figure 1 – Device:2 Data Model Structure – Overview

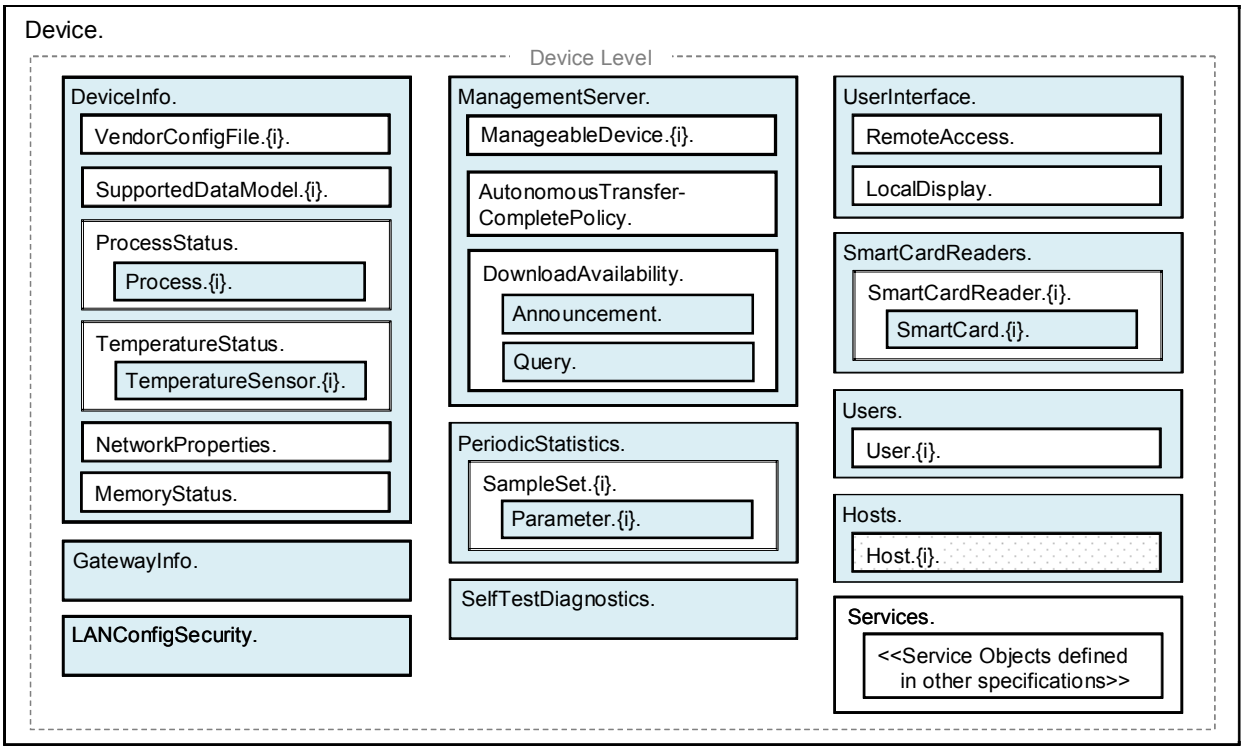

Figure 2 – Device:2 Data Model Structure – Device Level

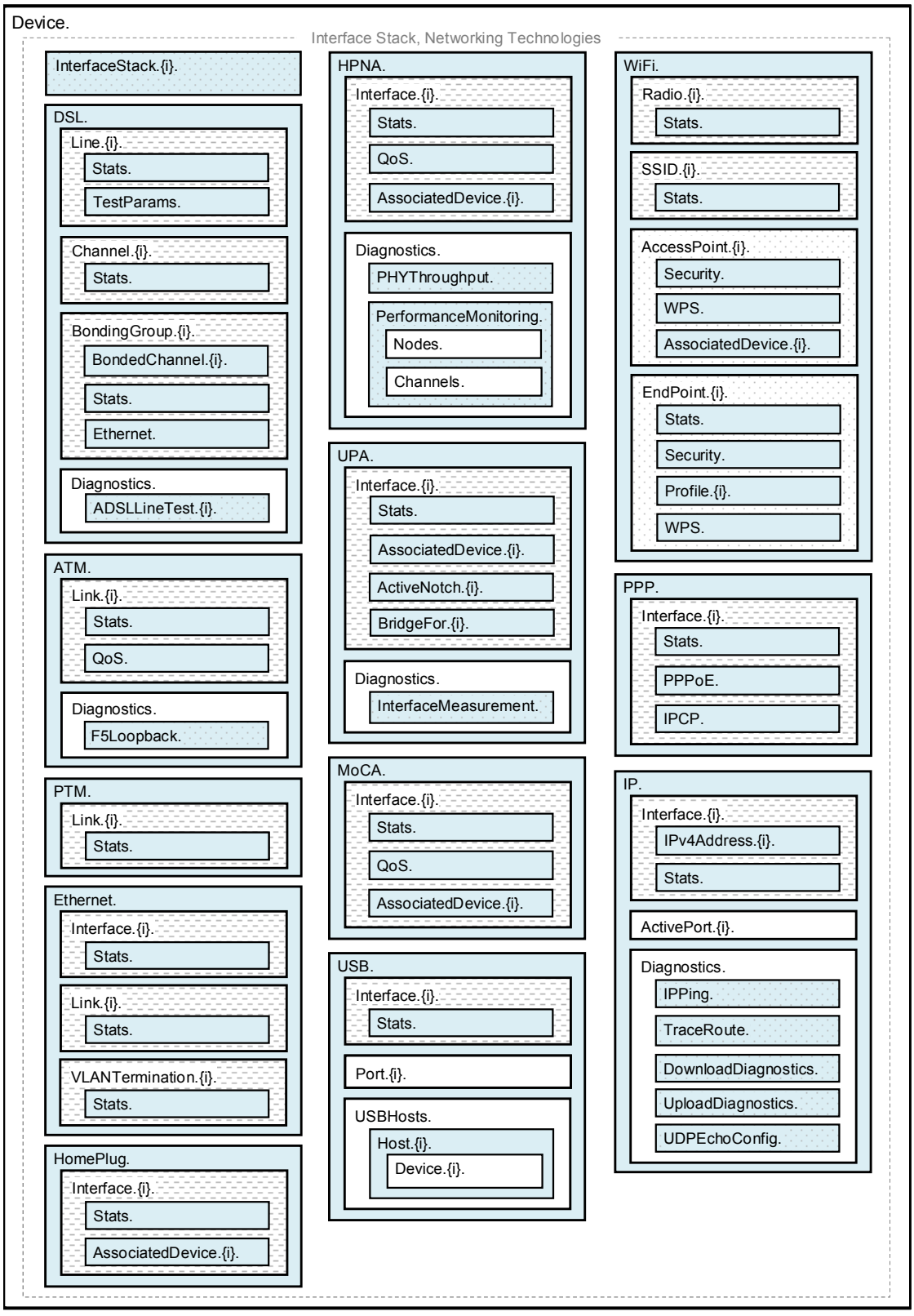

Figure 3 – Device:2 Data Model Structure – Interface Stack and Networking Technologies

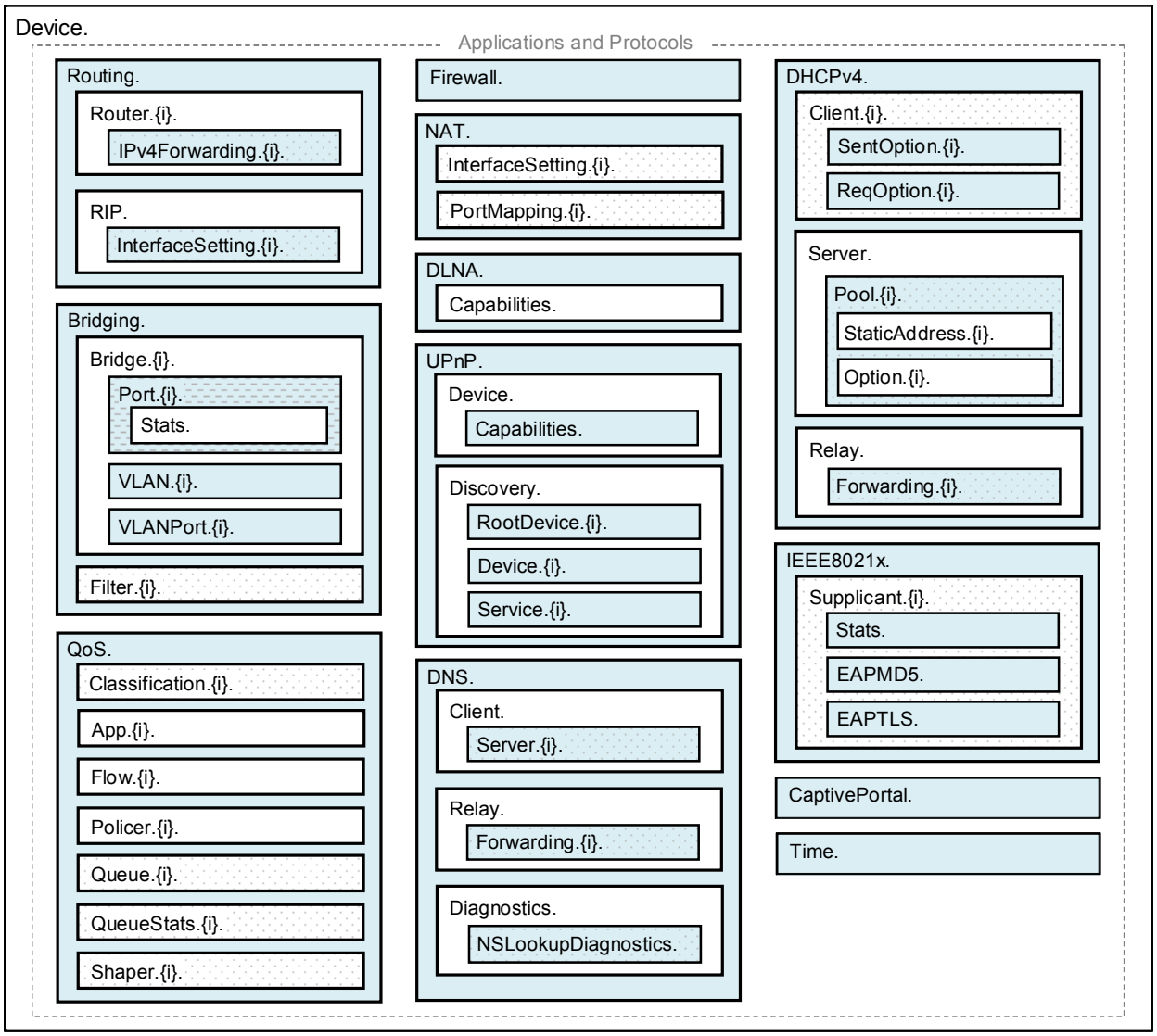

Figure 4 – Device:2 Data Model Structure – Applications and Protocols

2 References and Terminology

2.1 Conventions

In this Technical Report, several words are used to signify the requirements of the specification. These words are always capitalized. More information can be found be in RFC 2119 [1].

MUST	This word, or the terms "REQUIRED" or "SHALL", means that the definition is an absolute requirement of the specification.
MUST NOT	This phrase, or the phrase "SHALL NOT", means that the definition is an absolute prohibition of the specification.
SHOULD	This word, or the adjective "RECOMMENDED", means that there might exist valid reasons in particular circumstances to ignore this item, but the full implications need to be understood and carefully weighed before choosing a different course.
SHOULD NOT	This phrase, or the phrase "NOT RECOMMENDED" means that there might exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications need to be understood and the case carefully weighed before implementing any behavior described with this label.
MAY	This word, or the adjective "OPTIONAL", means that this item is one of an allowed set of alternatives. An implementation that does not include this option MUST be prepared to inter-operate with another implementation that does include the option.

The key words "DEPRECATED" and "OBSOLETED" in this Technical Report are to be interpreted as defined in TR-106 [3].

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.

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

[2] TR-069 Amendment 2, *CPE WAN Management Protocol*, Broadband Forum, 2007

[3] TR-106 Amendment 4, *Data Model Template for TR-069-Enabled Devices*, Broadband Forum, 2010

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

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

[6] RFC 2863, *The Interfaces Group MIB*, IETF, 2000

[7] X.200, Information technology - Open Systems Interconnection - Basic Reference Model: The basic model, ITU-T, 1994

[8] 802.1D-2004, Media Access Control (MAC) Bridges, IEEE, 2004

[9] 802.1Q-2005, Virtual Bridged Local Area Networks, IEEE, 2006

[10] RFC 2597, Assured Forwarding PHB Group, IETF, 1999

[11] RFC 3246, An Expedited Forwarding PHB (Per-Hop Behavior), IETF, 2002

[12] RFC 3261, SIP: Session Initiation Protocol, IETF, 2002

[13] RFC 3435, Media Gateway Control Protocol (MGCP) - Version 1.0, IETF, 2003

[14] RFC 4566, SDP: Session Description Protocol, IETF, 2006

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.
CPE	Customer Premises Equipment; refers to any TR-069-enabled [2] device and therefore covers Internet Gateway Devices, LAN-side End Devices, and other Network Infrastructure Devices.
Component	A named collection of *Objects* and/or *Parameters* and/or Profiles that can be included anywhere within a *Data Model*.
CWMP	*CPE* WAN Management Protocol. Defined in TR-069 [2], CWMP is a communication protocol between an *ACS* and *CPE* that defines a mechanism for secure auto-configuration of a *CPE* and other *CPE* management functions in a common framework.
Data Model	A hierarchical set of *Objects* and/or *Parameters* that define the managed objects accessible via TR-069 for a particular *CPE*.
Device	Used here as a synonym for *CPE*.
DM Instance	Data Model Schema instance document. This is an XML document that conforms to the *DM Schema* and to any additional rules specified in or referenced by the *DM Schema*.
DM Schema	Data Model Schema. This is the XML Schema [5] that is used for defining data models for use with *CWMP*.
Interface Object	A type of *Object* that models a network interface or protocol layer. Commonly referred to as an interface. They can be stacked, one on top of the other, using *Path References* in order to dynamically define the relationships between interfaces.
Object	A named collection of *Parameters* and/or other *Objects*.
Parameter	A name-value pair representing a manageable *CPE* parameter made accessible to an *ACS* for reading and/or writing.
Path Reference	Describes how a parameter can reference another parameter or object via its path name (Section A.2.3.4/TR-106 [3]). Such a reference can be weak or strong (Section A.2.3.6/TR-106 [3]).

2.4 Abbreviations

This Technical Report uses the following abbreviations:

ATM	Asynchronous Transfer Mode.
DSL	Digital Subscriber Line.
IP	Internet Protocol.
OSI	Open Systems Interconnection.
PPP	Point-to-Point Protocol.
PTM	Packet Transfer Mode.
RPC	Remote Procedure Call.
SSID	Service Set Identifier.
URI	Uniform Resource Identifier [4].
URL	Uniform Resource Locator [4].

3 Technical Report Impact

3.1 Energy Efficiency

TR-181 has no impact on Energy Efficiency.

3.2 IPv6

TR-181 does not specifically address IPv6 at this time, though the Device:2 data model is designed to be future proof. Future work is expected to define IPv6 extensions to the Device:2 data model.

3.3 Security

TR-181 has no impact on Security.

4 Architecture

4.1 Interface Layers

This Technical Report models network interfaces and protocol layers as independent data objects, generally referred to as interface objects (or interfaces). Interface objects can be stacked, one on top of the other, using path references in order to dynamically define the relationships between interfaces.

The interface object and interface stack are concepts inspired by RFC 2863 [6].

Within the Device:2 data model, interface objects are arbitrarily restricted to definitions that operate at or below the IP network layer (i.e. layers 1 through 3 of the OSI model [7]). However, vendor-specific interface objects MAY be defined which fall outside this restricted scope.

Figure 5 lists the interface objects defined in the Device:2 data model. The indicated OSI layer is non-normative; it serves as a guide only, illustrating at what level in the stack an interface object is expected to appear. However, a CPE need not support or use all interfaces, which means that the figure does not reflect all possible stacking combinations and restrictions. For example, one CPE stack might exclude DSL Bonding, while another CPE stack might include DSL Bonding but exclude Bridging, while still another might include VLANTermination under PPP, or VLANTermination under IP with no PPP, or even Ethernet Link under IP with no VLANTermination and no PPP.

NOTE – Throughout this Technical Report, 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 data model).

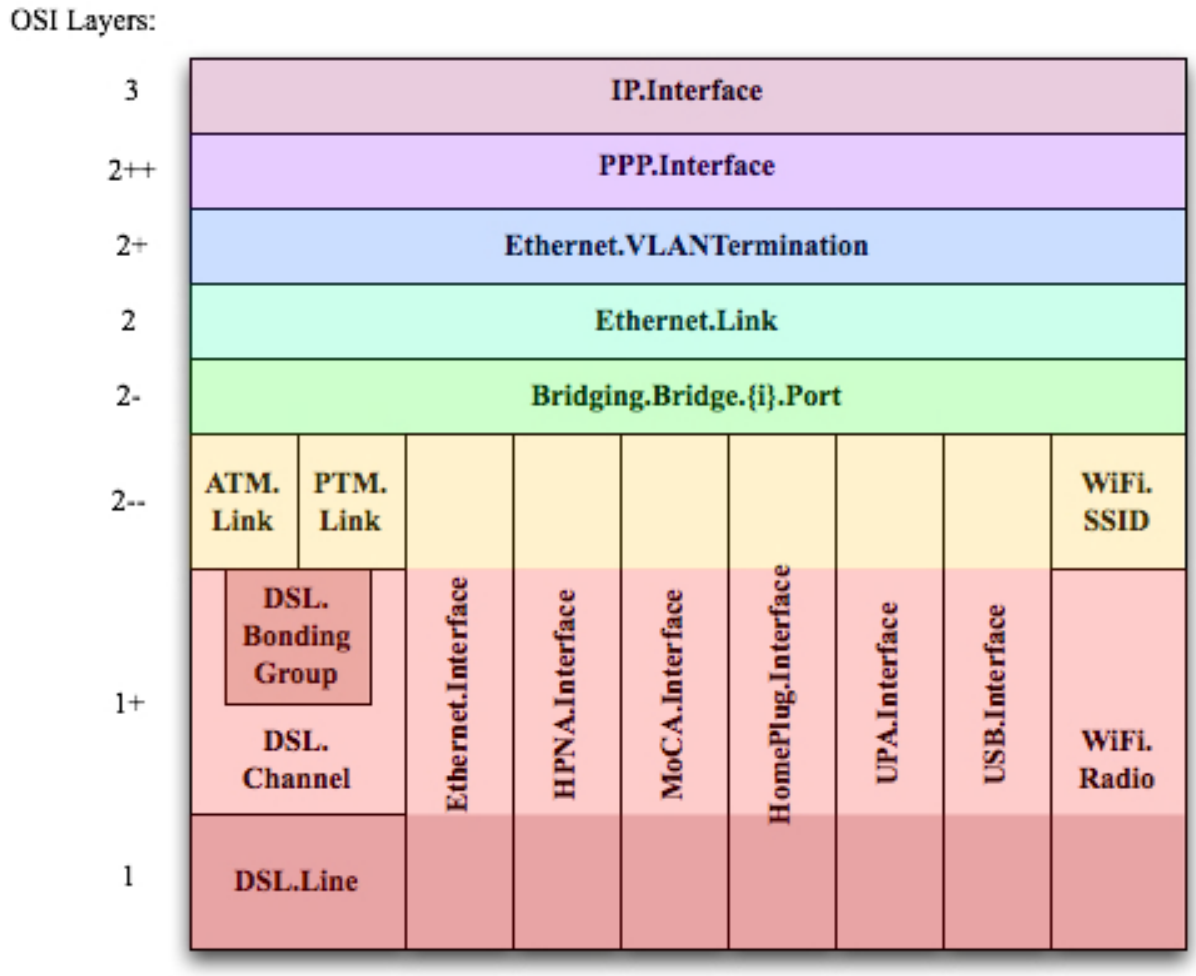

Figure 5 – OSI Layers and Interface Objects[1] [2]

4.2 Interface objects

An interface object is a type of network interface or protocol layer. Each type of interface is modeled by a Device:2 data model table, with a row per interface instance (e.g. IP.Interface.{i} for IP Interfaces).

Each interface object contains a core set of parameters and objects, which serves as the template for defining interface objects within the data model. Interface objects can also contain other parameters and sub-objects specific to the type of interface.

[1] Note that, because new minor versions of the Device:2 data model can be defined without re-publishing this Technical Report, the figure is not necessarily up-to-date.

[2] The Bridge.{i}.Port.{i} object models both management (upwards facing) Bridge Ports and non-management (downwards facing) Bridge Ports, where each instance is configured as one or the other. Management Bridge Ports are stacked above non-management Bridge Ports.

The core set of parameters consists of:

- Enable — The administrative state of the interface (i.e. boolean indicating enabled or disabled)
- Status — The operational state of the interface (i.e. Up, Down, Unknown, Dormant, NotPresent, LowerLayerDown, Error)
- Alias — An alternate name used to identify the interface, which is assigned an initial value by the CPE but can later be chosen by the ACS
- Name — The textual name used to identify the interface, which is chosen by the CPE
- LastChange — The accumulated time in seconds since the interface entered its current operational state
- LowerLayers — A list of path references to interface objects that are stacked immediately below the interface

Also, a core set of statistics parameters is contained within a Stats sub-object. The definition of these parameters MAY be customized for each interface type. The core set of parameters within the Stats sub-object consists of:

- BytesSent — The total number of bytes transmitted out of the interface, including framing characters.
- BytesReceived — The total number of bytes received on the interface, including framing characters.
- PacketsSent — The total number of packets transmitted out of the interface.
- PacketsReceived — The total number of packets received on the interface.
- ErrorsSent — The total number of outbound packets that could not be transmitted because of errors.
- ErrorsReceived — The total number of inbound packets that contained errors preventing them from being delivered to a higher-layer protocol.
- UnicastPacketsSent — The total number of packets requested for transmission which were not addressed to a multicast or broadcast address at this layer, including those that were discarded or not sent.
- UnicastPacketsReceived — The total number of received packets, delivered by this layer to a higher layer, which were not addressed to a multicast or broadcast address at this layer.
- DiscardPacketsSent — The total number of outbound packets which were chosen to be discarded even though no errors had been detected to prevent their being transmitted.

- DiscardPacketsReceived — The total number of inbound packets which were chosen to be discarded even though no errors had been detected to prevent their being delivered.
- MulticastPacketsSent — The total number of packets that higher-layer protocols requested for transmission and which were addressed to a multicast address at this layer, including those that were discarded or not sent.
- MulticastPacketsReceived — The total number of received packets, delivered by this layer to a higher layer, which were addressed to a multicast address at this layer.
- BroadcastPacketsSent — The total number of packets that higher-level protocols requested for transmission and which were addressed to a broadcast address at this layer, including those that were discarded or not sent.
- BroadcastPacketsReceived — The total number of received packets, delivered by this layer to a higher layer, which were addressed to a broadcast address at this layer.
- UnknownProtoPackets-Received — The total number of packets received via the interface, which were discarded because of an unknown or unsupported protocol.

NOTE – The CPE MUST reset an interface's Stats parameters (unless otherwise stated in individual object or parameter descriptions) either when the interface becomes operationally down due to a previous administrative down (i.e. the interface's Status parameter transitions to a down state after the interface is disabled) or when the interface becomes administratively up (i.e. the interface's Enable parameter transitions from *false* to *true*). Administrative and operational status is discussed in Section 4.2.2.

4.2.1 Lower Layers

Each interface object can be stacked on top of zero or more other interface objects, which MUST be specified using its LowerLayers parameter. By having each interface object, in turn, reference the interface objects in its lower layer, a logical hierarchy of all interface relationships is built up.

The LowerLayers parameter is a comma-separated list of path references to interface objects. Each item in the list represents an interface object that is stacked immediately below the referencing interface. If a referenced interface is deleted, the CPE MUST remove the corresponding item from this list (i.e. items in the LowerLayers parameter are strong references).

These relationships between interface objects can either be set by management action, in order to specify new interface configurations, or be pre-configured within the CPE.

A CPE MUST reject any attempt to set LowerLayers values that would result in an invalid or unsupported configuration. The corresponding fault response from the CPE MUST indicate this using an Invalid Parameter Value fault code (9007). See Section A.3.2.1/TR-069 [2] for further details on SetParameterValues fault responses.

The lowest layer in a fully configured and operational stack is generally the physical interface (e.g. DSL Line instance representing a DSL physical link). Within these physical interface objects the LowerLayers parameter will be an empty list, unless some lower layer vendor-specific interface objects are defined and present. Higher layer interface objects MAY operate without a physical layer being modeled, however this is a local matter to the CPE.

Figure 6 illustrates the use of the LowerLayers parameter. A, B, C, and D represent interface objects. Interface A's LowerLayers parameter references interfaces B and C. Interface B's LowerLayers parameter references interface D. Interfaces C and D have no interface references specified in their LowerLayers parameters. In this way, a multi-layered interface stack is configured. If the ACS were to delete interface B, then the CPE would update interface A's LowerLayers parameter to no longer reference interface B (and interface D would be stranded, no longer referenced by the now deleted interface B).

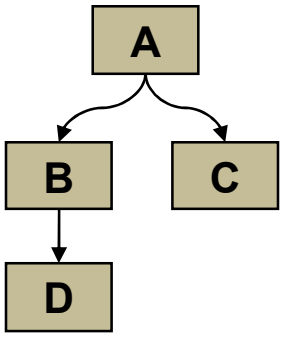

Figure 6 – Interface LowerLayers

4.2.2 Administrative and Operational Status

NOTE – Many of the requirements outlined in this section were derived from Section 3.1.13/RFC 2863 [6].

An interface object's Enable and Status parameters specify the current administrative and operational status of the interface, respectively. Valid values for the Status parameter are: Up, Down, Unknown, Dormant, NotPresent, LowerLayerDown, and Error.

The CPE MUST do everything possible in order to follow the operational state transitions as described below. In some cases these requirements are defined as SHOULD; this is not an indication that they are optional. These transitions, and the relationship between the Enable parameter and the Status parameter, are required behavior – it is simply the timing of how long these state transitions take that is implementation specific.

When the Enable parameter is *false* the Status parameter SHOULD normally be *Down (or NotPresent or Error if there is a fault condition on the interface)*. Note that when the Enable parameter transitions to *false*, it is possible that the Status parameter's transition to *Down* might occur after a small time lag if the CPE needs to first complete certain operations (e.g. finish transmitting a packet).

When the Enable parameter is changed to *true*, the Status SHOULD do one of the following:

- Change to *Up* if and only if the interface is able to transmit and receive network traffic.
- Change to *Dormant* if and only if the interface is operable, but is waiting for external actions before it can transmit and receive network traffic.
- Change to *LowerLayerDown* if and only if the interface is prevented from entering the *Up* state because one or more of the interfaces beneath it is down.
- Remain in the *Error* state if there is an error or other fault condition detected on the interface.
- Remain in the *NotPresent* state if the interface has missing (typically hardware) components.
- Change to *Unknown* if the state of the interface can not be determined for some reason.

The *Dormant* state indicates that the interface is operable, but it is waiting for external events to occur before it can transmit/receive traffic. When such events occur, and the interface is then able to transmit/receive traffic, the Status SHOULD change to the *Up* state. Note that both the *Up* and *Dormant* states are considered healthy states.

The *Down*, *NotPresent*, *LowerLayerDown,* and *Error* states all indicate that the interface is down. The *NotPresent* state indicates that the interface is down specifically because of a missing (typically hardware) component. The *LowerLayerDown* state indicates that the interface is stacked on top of one or more other interfaces, and that this interface is down specifically because one or more of these lower-layer interfaces is down.

The *Error* state indicates that the interface is down because an error or other fault condition was detected on the interface.

4.2.3 Stacking and Operational Status

NOTE – The requirements outlined in this section were derived from Section 3.1.14/RFC 2863 [6].

When an interface object is stacked on top of lower-layer interfaces (i.e. is not a bottommost layer in the stack), then:

- The interface SHOULD be *Up* if it is able to transmit/receive traffic due to one or more interfaces lower down in the stack being *Up*, irrespective of whether other interfaces below it are in a non-*Up* state (i.e. the interface is functioning in conjunction with at least some of its lower-layered interfaces).
- The interface MAY be *Up* or *Dormant* if one or more interfaces lower down in the stack are *Dormant* and all other interfaces below it are in a non-*Up* state.
- The interface is expected to be *LowerLayerDown* while all interfaces lower down in the stack are either *Down*, *NotPresent*, *LowerLayerDown,* or *Error*.

4.2.4 Vendor-specific Interface Objects

Vendor-specific interface objects MAY be defined and used. If such objects are specified by vendors, they MUST be preceded by *X_<VENDOR>_* and follow the syntax for vendor extensions used for parameter names (as defined in Section 3.3/TR-106 [3]).

If the ACS encounters an unknown vendor-specific interface object within a CPE's interface stack, rather than responding with a fault, the ACS MUST proceed as if this object's upper-layer interfaces were directly linked to its lower-layer interfaces. This applies whether the ACS encounters such an object via the InterfaceStack table (Section 4.3) or via an interface object's LowerLayers parameter.

Figure 7 illustrates a stacked vendor-specific interface object being bypassed by the ACS, where there is just one object below the vendor-specific object.

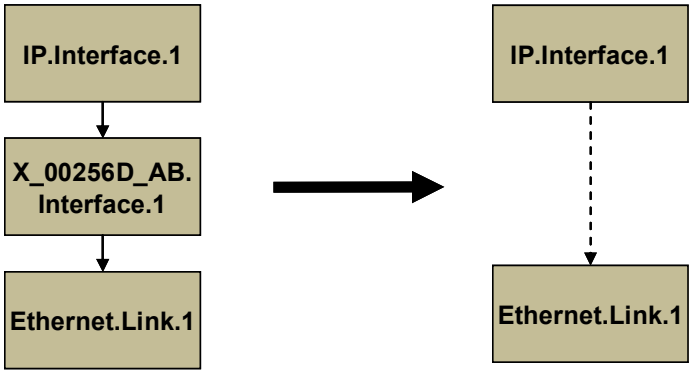

Figure 7 – Ignoring a Vendor-specific Interface Object in the Stack

Figure 8 illustrates a stacked vendor-specific interface object being bypassed by the ACS, where there are multiple objects below the vendor-specific object.

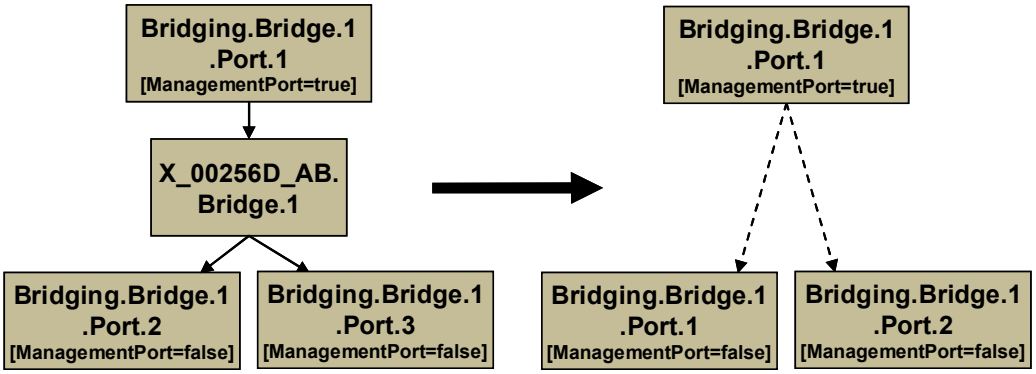

Figure 8 – Ignoring a Vendor-specific Interface Object in the Stack (multiple sub-objects)

4.3 InterfaceStack Table

Although the interface stack can be traversed via LowerLayers parameters (as described in Section 4.2.1 *Lower Layers*), an alternate mechanism is provided to aid in visualizing the overall stacking relationships and to quickly access objects within the stack.

The InterfaceStack table is a Device:2 data model object, namely *Device.InterfaceStack.{i}*. This is a read-only table whose rows are auto-generated by the CPE based on the current relationships that are configured between interface objects (via each interface instance's LowerLayers parameter). Each table row represents a "link" between a higher-layer interface object

(referenced by its HigherLayer parameter) and a lower-layer interface object (referenced by its LowerLayer parameter). This means that an InterfaceStack table row's HigherLayer and LowerLayer parameters will always both be non-null.

NOTE – As a consequence, interface instances that have been stranded will not be represented within the InterfaceStack table[3]. It is also likely that multiple, disjoint groups of stacked interface objects will coexist within the table (for example, each IP interface will be the root of a disjoint group; unused "fragments", e.g. a secondary DSL channel with a configured ATM PVC that isn't attached to anything above, will linger if they remain interconnected; and finally, partially configured "fragments" can be present when an interface stack is being set up).

A CPE MUST autonomously add or remove rows in the InterfaceStack table in response to the following circumstances:

- An interface's LowerLayers parameter was updated to remove a reference to another interface (i.e. a "link" is being removed from the stack due to a SetParameterValues request).
- An interface's LowerLayers parameter was updated to add a reference to another interface (i.e. a "link" is being added to the stack due to a SetParameterValues request).
- An interface was deleted that had referenced, or been referenced by, one other interface (i.e. a "link" is being removed from the stack due to a DeleteObject request).
- An interface was deleted that had referenced, or been referenced by, multiple interfaces (i.e. multiple "links" are being removed from the stack due to a DeleteObject request).

Once the CPE issues the SetParameterValuesResponse or the DeleteObjectResponse, all autonomous InterfaceStack table changes associated with the corresponding request (as described in the preceding paragraph) MUST be available for subsequent commands to operate on, regardless of whether or not these changes have been applied by the CPE (see TR-069 [2] Sections A.3.2.1 and A.3.2.7 for background on these RPC methods).

As an example, Table 1 lists an InterfaceStack table configuration imagined for a fictitious, simple router. Each row in this table corresponds to a row in the InterfaceStack table. The specified objects and instance numbers are manufactured for the sake of this example; real world configurations will likely differ.

Table 1 – Simple Router Example (InterfaceStack table)

Row/Instance	Higher Layer Interface	Lower Layer Interface
1	IP.Interface.1	PPP.Interface.1
2	PPP.Interface.1	Ethernet.Link.1
3	Ethernet.Link.1	ATM.Link.1
4	ATM.Link.1	DSL.Channel.1
5	DSL.Channel.1	DSL.Line.1
6	IP.Interface.2	Ethernet.Link.2

[3] An interface instance is considered stranded when it has no lower layer references to or from other interface instances. Stranded interface instances will be omitted from the InterfaceStack table until such time as they are stacked, above or below another interface instance, via a LowerLayers parameter reference.

Row/Instance	Higher Layer Interface	Lower Layer Interface
7	Ethernet.Link.2	ATM.Link.2
8	ATM.Link.2	DSL.Channel.1
9	IP.Interface.3	Ethernet.Link.3
10	Ethernet.Link.3	Bridging.Bridge.1.Port.1
11	Bridging.Bridge.1.Port.1	Bridging.Bridge.1.Port.2
12	Bridging.Bridge.1.Port.2	Ethernet.Interface.1
13	Bridging.Bridge.1.Port.1	Bridging.Bridge.1.Port.3
14	Bridging.Bridge.1.Port.3	Ethernet.Interface.2
15	Bridging.Bridge.1.Port.1	Bridging.Bridge.1.Port.4
16	Bridging.Bridge.1.Port.4	WiFi.SSID.1
17	WiFi.SSID.1	WiFi.Radio.1

By looking at the rows from the example InterfaceStack table as a whole, we can visualize the overall stack configuration. Figure 9 shows how this information can be pictured. Interface instances are represented by colored boxes, while InterfaceStack instances are represented by numbered circles.

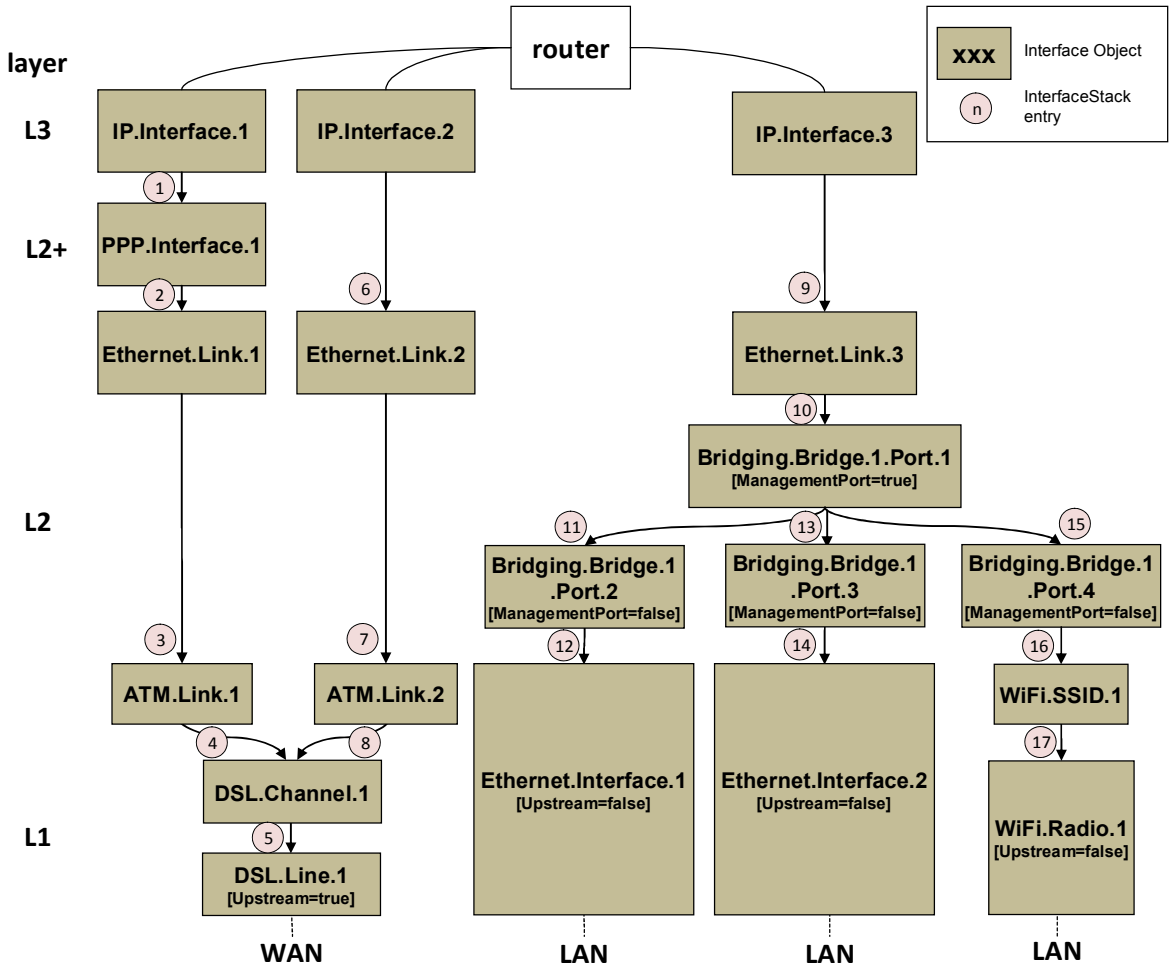

Figure 9 – Simple Router Example (Interfaces Visualized)

Finally, Table 2 completes the example by listing each interface instance and its corresponding LowerLayers parameter value.

Table 2 – Simple Router Example (Interface LowerLayers)

Interface	LowerLayers value
IP.Interface.1	PPP.Interface.1
IP.Interface.2	Ethernet.Link.2
IP.Interface.3	Ethernet.Link.3
PPP.Interface.1	Ethernet.Link.1
Ethernet.Link.1	ATM.Link.1
Ethernet.Link.2	ATM.Link.2
Ethernet.Link.3	Bridging.Bridge.1.Port.1
Bridging.Bridge.1.Port.1	Bridging.Bridge.1.Port.2, Bridging.Bridge.1.Port.3, Bridging.Bridge.1.Port.4
Bridging.Bridge.1.Port.2	Ethernet.Interface.1
Bridging.Bridge.1.Port.3	Ethernet.Interface.2
Bridging.Bridge.1.Port.4	WiFi.SSID.1
ATM.Link.1	DSL.Channel.1
ATM.Link.2	DSL.Channel.1
DSL.Channel.1	DSL.Line.1
DSL.Line.1	
Ethernet.Interface.1	
Ethernet.Interface.2	
WiFi.SSID.1	WiFi.Radio.1
WiFi.Radio.1	

5 Parameter Definitions

The normative definition of the Device:2 data model is split between several DM Instance documents (see TR-069 [2] Annex A). Table 3 lists the Device:2 data model versions and DM Instances that had been defined at the time of writing. It also indicates the corresponding Technical Reports and gives links to the associated XML and HTML files. The TR-181i2 XML document defines the Device:2 model itself, and imports additional components from the other XML documents listed. The TR-181i2 HTML document is a report generated from the XML files, and lists the entire Device:2 data model in human-readable form.

Note that, because new minor versions of the Device:2 data model can be defined without re-publishing this Technical Report, the table is not necessarily up-to-date. An up-to-date version of this information can always be found at http://www.broadband-forum.org/cwmp.

Table 3 – Device:2 Data Model Versions

Version	DM Instance	Technical Report	XML and HTML
2.0	tr-181-2-0.xml	TR-181 Issue 2	http://broadband-forum.org/cwmp/tr-181-2-0.xml
			http://broadband-forum.org/cwmp/tr-181-2-0.html
	tr-143-1-0.xml[4]	TR-143	http://broadband-forum.org/cwmp/tr-143-1-0.xml
	tr-157-1-2.xml	TR-157 Amendment 2	http://broadband-forum.org/cwmp/tr-157-1-2.xml

[4] The minimum valid version of the tr-143-1-0.xml document is corrigendum 2. Earlier versions are not supported by the Device:2 data model.

Annex A: Bridging and Queuing

A.1 Queuing and Bridging Model

Figure 10 shows the queuing and bridging model for a device. This model relates to the QoS object as well as the Bridging and Routing objects. The elements of this model are described in the following sections.

NOTE – the queuing model described in this Annex is meant strictly as a model to clarify the intended behavior of the related data objects. There is no implication intended that an implementation has to be structured to conform to this model.

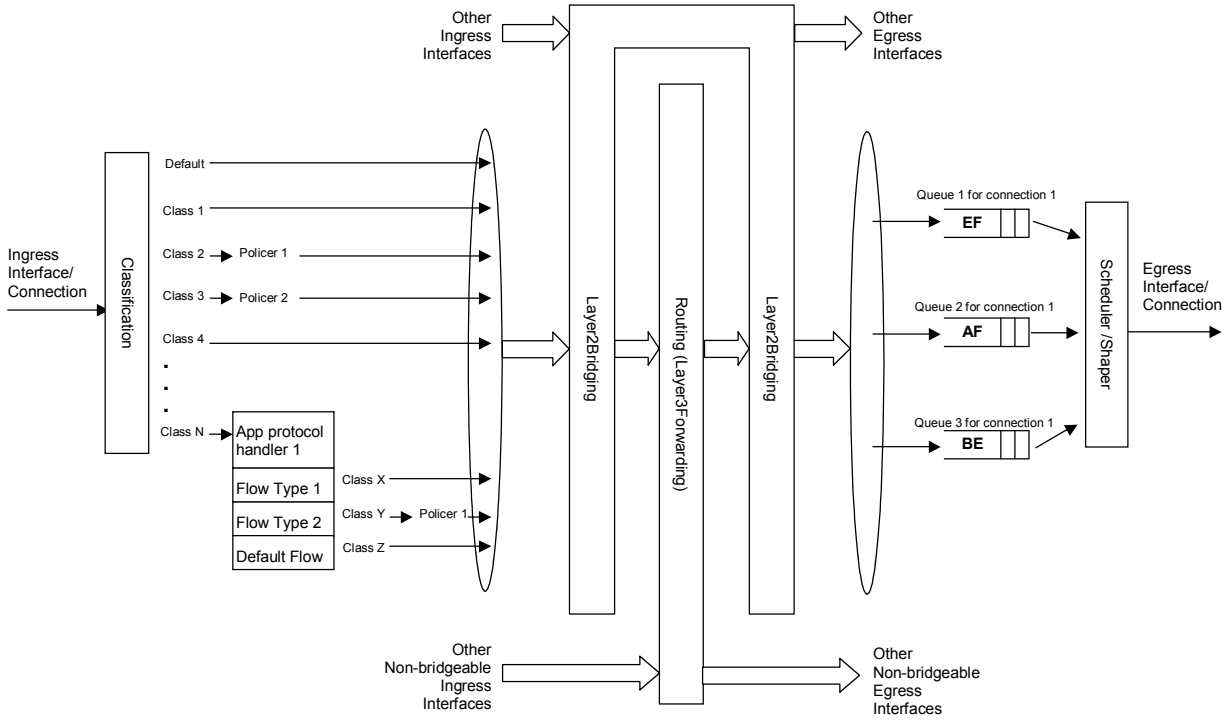

Figure 10 – Queuing Model of a Device

A.1.1 Packet Classification

The Classification table within the QoS object specifies the assignment of each packet arriving at an ingress interface to a specific internal class. This classification can be based on a number of matching criteria, such as destination and source IP address, destination and source port, and protocol.

Each entry in the Classification table includes a series of parameters, each indicated to be a Classification Criterion. Each classification criterion can be set to a specified value, or can be set to a value that indicates that criterion is not to be used. A packet is defined to match the classification criteria for that table entry only if the packet matches all of the specified criteria. That is, a logical AND operation is applied across all classification criteria within a given Classification table entry.

NOTE – to apply a logical OR to sets of classification criteria, multiple entries in the Classification table can be created that specify the same resulting queuing behavior.

For each classification criterion, the Classification table also includes a corresponding "exclude" flag. This flag can be used to invert the sense of the associated classification criterion. That is, if this flag is *false* for a given criterion, the classifier is to include only packets that meet the specified criterion (as well as all others). If this flag is *true* for a given criterion, the classifier is to include all packets except those that meet the associated criterion (in addition to meeting all other criteria).

For a given entry in the Classification table, the classification is to apply only to the interface specified by the Interface parameter. This parameter can specify a particular ingress interface or all sources. Depending on the particular interface, not all classification criteria will be applicable. For example, Ethernet layer classification criteria would not apply to packets arriving on a non-bridged ATM VC.

Packet classification is modeled to include all ingress packets regardless of whether they ultimately will be bridged or routed through the device.

A.1.1.1 Classification Order

The class assigned to a given packet corresponds to the first entry in the Classification table (given the specified order of the entries in the table) whose matching criteria match the packet. If there is no entry that matches the packet, the packet is assigned to a default class.

Classification rules are sensitive to the order in which they are applied because certain traffic might meet the criteria of more than one Classification table entry. The Order parameter is responsible for identifying the order in which the Classification entries are to be applied.

The following rules apply to the use and setting of the Order parameter:

- Order goes in order from 1 to n, where n is equal to the number of entries in the Classification table. 1 is the highest precedence, and n the lowest. For example, if entries with Order of 4 and 7 both have rules that match some particular traffic, the traffic will be classified according to the entry with the 4.
- The CPE is responsible for ensuring that all Order values are unique and sequential.
    - If an entry is added (number of entries becomes n+1), and the value specified for Order is greater than n+1, then the CPE will set Order to n+1.
    - If an entry is added (number of entries becomes n+1), and the value specified for Order is less than n+1, then the CPE will create the entry with that specified value, and increment the Order value of all existing entries with Order equal to or greater than the specified value.
    - If an entry is deleted, the CPE will decrement the Order value of all remaining entries with Order greater than the value of the deleted entry.
    - If the Order value of an entry is changed, then the value will also be changed for other entries greater than or equal to the lower of the old and new values, and less than the

larger of the old and new values. If the new value is less than the old, then these other entries will all have Order incremented. If the new value is greater than the old, then the other entries will have Order decremented and the changed entry will be given a value of <new value>-1. For example, an entry is changed from 8 to 5. The existing 5 goes to 6, 6 to 7, and 7 to 8. If the entry goes from 5 to 8, then 6 goes to 5, 7 to 6, and the changed entry is 7. This is consistent with the behavior that would occur if the change were considered to be an Add of a new entry with the new value, followed by a Delete of the entry with the old value.

A.1.1.2 Dynamic Application Specific Classification

In some situations, traffic to be classified cannot be identified by a static set of classification criteria. Instead, identification of traffic flows might require explicit application awareness. The model accommodates such situations via the App and Flow tables in the QoS object.

Each entry in the App table is associated with an application-specific protocol handler, identified by the ProtocolIdentifier, which contains a URN. For a particular CPE, the AvailableAppList parameter indicates which protocol handlers that CPE is capable of supporting, if any. A list of standard protocol handlers and their associated URNs is specified in Section A.3, though a CPE can also support vendor-specific protocol handlers as well. Multiple App table entries can refer to the same ProtocolIdentifier.

The role of the protocol handler is to identify and classify flows based on application awareness. For example, a SIP protocol handler might identify a call-control flow, an audio flow, and a video flow. The App and Flow tables are used to specify the classification outcome associated with each such flow.

For each App table entry there can be one or more associated Flow table entries. Each flow table entry identifies a type of flow associated with the protocol handler. The Type parameter is used to identify the specific type of flow associated with each entry. For example, a Flow table entry for a SIP protocol handler might refer only to the audio flows associated with that protocol handler. A list of standard flow type values is given in Section A.3, though a CPE can also support vendor-specific flow types.

A protocol handler can be defined as being fed from the output of a Classification table entry. That is, a Classification entry can be used to single out control traffic to be passed to the protocol handler, which then subsequently identifies associated flows. Doing so allows more than one instance of a protocol handler associated with distinct traffic. For example, one could define two App table entries associated with SIP protocol handlers. If the classifier distinguished control traffic to feed into each handler based on the destination IP address of the SIP server, this could be used to separately classify traffic for different SIP service providers. In this case, each instance of the protocol handler would identify only those flows associated with a given service. Note that the Classification table entry that feeds each protocol handler wouldn't encompass all of the flows; only the traffic needed by the protocol handler to determine the flows—typically only the control traffic.

A.1.1.3 Classification Outcome

Each Classification entry specifies a tuple composed of either:

- A TrafficClass and (optionally) a Policer, or
- An App table entry

Each entry also specifies:

- Outgoing DiffServ and Ethernet priority marking behavior
- A ForwardingPolicy tag that can be referenced in the Routing table to affect packet routing (note that the ForwardingPolicy tag affects only routed traffic)

Note that the information associated with the classification outcome is modeled as being carried along with each packet as it flows through the system.

If a packet does not match any Classification table entry, the DefaultTrafficClass, DefaultPolicer, default markings, and default ForwardingPolicy are used.

If a TrafficClass/Policer tuple is specified, classification is complete. If, however, an App is specified, the packet is passed to the protocol handler specified by the ProtocolIdentifier in the specified App table entry for additional classification (see Section A.1.1.2). If any of the identified flows match the Type specified in any Flow table entry corresponding to the given App table entry (this correspondence is indicated by the App identifier), the specified tuple and markings for that Flow table entry is used for packets in that flow. Other flows associated with the application, but not explicitly identified, use the default tuple and markings specified for that App table entry.

A.1.2 Policing

The Policer table defines the policing parameters for ingress packets identified by either a Classification table entry (or the default classification) or a dynamic flow identified by a protocol handler identified in the App table.

Each Policer table entry specifies the packet handling characteristics, including the rate requirements and behavior when these requirements are exceeded.

A.1.3 Queuing and Scheduling

The Queue table specifies the number and types of queues, queue parameters, shaping behavior, and scheduling algorithm to use. Each Queue table entry specifies the TrafficClasses with which it is associated, and a set of egress interfaces for which a queue with the corresponding characteristics needs to exist.

NOTE – If the CPE can determine that among the interfaces specified for a queue to exist, packets classified into that queue cannot egress to a subset of those interfaces (from knowledge of the current routing and bridging configuration), the CPE can choose not to instantiate the queue on those interfaces.

NOTE – Packets classified into a queue that exit through an interface for which the queue is not specified to exist, will instead use the default queuing behavior. The default queue itself will exist on all egress interfaces.

The model defined here is not intended to restrict where the queuing is implemented in an actual implementation. In particular, it is up to the particular implementation to determine at what protocol layer it is most appropriate to implement the queuing behavior (IP layer, Ethernet MAC layer, ATM layer, etc.). In some cases, however, the QoS configuration would restrict the choice of layer where queueing can be implemented. For example, if a queue is specified to carry traffic that is bridged, then it could not be implemented as an IP-layer queue.

NOTE – care needs to be taken to avoid having multiple priority queues multiplexed onto a single connection that is rate shaped. In such cases, the possibility exists that high priority traffic can be held back due to rate limits of the overall connection exceeded by lower priority traffic. Where possible, each priority queue will be shaped independently using the shaping parameters in the Queue and Shaping table.

The scheduling parameters defined in the Queue table apply to the first level of what might be a more general scheduling hierarchy. This specification does not specify the rules that an implementation needs to apply to determine the most appropriate scheduling hierarchy given the scheduling parameters defined in the Queue table.

As an example, take a situation where the output of four distinct queues is to be multiplexed into a single connection, and two entries share one set of scheduling parameters while the other two entries share a different set of scheduling parameters. In this case, it might be appropriate to implement this as a scheduling hierarchy with the first two queues multiplexed with a scheduler defined by the first pair, and the second two queues being multiplexed with a scheduler defined by the second pair. The lower layers of this scheduling hierarchy cannot be directly determined from the content of the Queue table.

A.1.4 Bridging

NOTE – from the point of view of a bridge, packets arriving into the bridge from the local router (either LAN-side or WAN-side) are treated as ingress packets, even though the same packets, which just left the router, are treated as egress from the point of view of the router. For example, a Filter table entry might admit packets on ingress to the bridge from a particular IP interface, which means that it admits packets on their way out of the router over this layer 3 connection.

For each interface, the output of the classifier is modeled to feed a set of 802.1D [8] or 802.1Q [9] layer 2 bridges as specified by the Bridging object. Each bridge specifies layer 2 connectivity between one or more layer 2 LAN and/or WAN interfaces, and optionally one or more layer 3 connections to the local router.

Each bridge corresponds to a single entry in the Bridge table of the Bridging object. The Bridge table contains the following sub-tables:

- **Port table**: models the Bridge ports, which are either management ports (modeling layer 3 connections to the local router) or non-management ports (modeling connections to layer 2 interfaces). Bridge ports are stackable interface objects (see Section 4.2).
- **VLAN table**: models the Bridge VLANs (relevant only to 802.1Q bridges).
- **VLANPort table**: for each VLAN, defines the ports that comprise its member set (relevant only to 802.1Q bridges).

A.1.4.1 Filtering

Traffic from a given interface (or set of interfaces) can be selectively admitted to a given Bridge, rather than bridging all traffic from that interface. Each entry in the Filter table includes a series of classification criteria. Each classification criterion can be set to a specified value, or can be set to a value that indicates that criterion is not to be used. A packet is admitted to the Bridge only if the packet matches all of the specified criteria. That is, a logical AND operation is applied across all classification criteria within a given Filter table entry.

NOTE – to apply a logical OR to sets of classification criteria, multiple entries in the Filter table can be created that refer to the same interfaces and the same Bridge table entry.

NOTE – a consequence of the above rule is that, if a packet does not match the criteria of any of the enabled Filter table entries, then it will not be admitted to any bridges, i.e. it will be dropped. As a specific example of this, if none of the enabled Filter table entries reference a given interface, then all packets arriving on that interface will be dropped.

For each classification criterion, the Filter table also includes a corresponding "exclude" flag. This flag can be used to invert the sense of the associated classification criterion. That is, if this flag is *false* for a given criterion, the Bridge will admit only packets that meet the specified criterion (as well as all other criteria). If this flag is *true* for a given criterion, the Bridge will admit all packets except those that meet the associated criterion (in addition to meeting all other criteria).

Note that because the classification criteria are based on layer 2 packet information, if the selected port for a given Filter table entry is a layer 3 connection from the local router, the layer 2 classification criteria do not apply.

A.1.4.2 Filter Order

Any packet that matches the filter criteria of one or more filters is admitted to the Bridge associated with the first entry in the Filter table (relative to the specified Order).

The following rules apply to the use and setting of the Order parameter:

- The Order goes in order from 1 to n, where n is equal to the number of filters. 1 is the highest precedence, and n the lowest.
- The CPE is responsible for ensuring that all Order values among filters are unique and sequential.
    - If a filter is added (number of filters becomes n+1), and the value specified for Order is greater than n+1, then the CPE will set Order to n+1.
    - If a filter is added (number of entries becomes n+1, and the value specified for Order is less than n+1, then the CPE will create the entry with that specified value, and increment the Order value of all existing filters with Order equal to or greater than the specified value.
    - If a filter is deleted, the CPE will decrement the Order value of all remaining filters with Order greater than the value of the deleted entry.

- If the Order value of a filter is changed, then the value will also be changed for other filters greater than or equal to the lower of the old and new values, and less than the larger of the old and new values. If the new value is less than the old, then these other entries will all have Order incremented. If the new value is greater than the old, then the other entries will have Order decremented and the changed entry will be given a value of <new value>-1. For example, an entry is changed from 8 to 5. The existing 5 goes to 6, 6 to 7, and 7 to 8. If the entry goes from 5 to 8, then 6 goes to 5, 7 to 6, and the changed entry is 7. This is consistent with the behavior that would occur if the change were considered to be an Add of a new filter with the new value, followed by a Delete of the filter with the old value.

A.2 Default Layer 2/3 QoS Mapping

Table 4 presents a "default" mapping between layer 2 and layer 3 QoS. In practice, it is a guideline for automatic marking of DSCP (layer 3) based upon Ethernet Priority (layer 2) and the other way around. Please refer to the QoS Classification table's DSCPMark and EthernetPriorityMark parameters (and related parameters) for configuration of a default automatic DSCP / Ethernet Priority mapping.

Automatic marking of DSCP or Ethernet Priority is likely only in the following cases:

- WAN → LAN: to map DSCP (layer 3) to Ethernet Priority (layer 2)
- LAN → WAN: to map Ethernet Priority (layer 2) to DSCP (layer 3)

Automatic marking in the LAN → LAN case is unlikely, since LAN QoS is likely to be supported only at layer 2, and LAN DSCP values, if used, will probably be a direct representation of Ethernet Priority, e.g. Ethernet Priority shifted left by three bits.

In the table, grayed and bolded items are added to allow two-way mapping between layer 2 and layer 3 QoS (where the mapping is ambiguous, the grayed values SHOULD be ignored and the bolded values SHOULD be used). If, when mapping from layer 3 to layer 2 QoS, the DSCP value is not present in the table, the mapping SHOULD be based only on the first three bits of the DSCP value, i.e. on DSCP & 111000.

Table 4 – Default Layer 2/3 QoS Mapping

Layer 2		Layer 3	
Ethernet Priority	Designation	DSCP	Per Hop Behavior
001 (1)	BK	000000 (0x00)	Default
010 (2)	spare	000000 (0x00)	
000 (0)	BE	000000 (0x00) **000000 (0x00)**	Default CS0
011 (3)	EE	001110 (0x0e) 001100 (0x0c) 001010 (0x0a) **001000 (0x08)**	AF13 AF12 AF11 CS1
100 (4)	CL	010110 (0x16) 010100 (0x14) 010010 (0x12) **010000 (0x10)**	AF23 AF22 AF21 CS2

101 (5)	VI	011110 (0x1e) 011100 (0x1c) 011010 (0x1a) **011000 (0x18)**	AF33 AF32 AF31 CS3
110 (6)	VO	100110 (0x26) 100100 (0x24) 100010 (0x22) **100000 (0x20)**	AF43 AF42 AF41 CS4
110 (6)	VO	101110 (0x2e) **101000 (0x28)**	EF CS5
111 (7)	NC	110000 (0x30) **111000 (0x38)**	CS6 CS7

A.3 URN Definitions for App and Flow Tables

A.3.1 App ProtocolIdentifier

Table 5 lists the URNs defined for the QoS App table's ProtocolIdentifier parameter. Additional standard or vendor-specific URNs can be defined following the standard syntax for forming URNs.

Table 5 – ProtocolIdentifer URNs

URN	Description
urn:dslforum-org:sip	Session Initiation Protocol (SIP) as defined by RFC 3261 [12]
urn:dslforum-org:h.323	ITU-T Recommendation H.323
urn:dslforum-org:h.248	ITU-T Recommendation H.248 (MEGACO)
urn:dslforum-org:mgcp	Media Gateway Control Protocol (MGCP) as defined by RFC 3435 [13]
urn:dslforum-org:pppoe	Bridged sessions of PPPoE

A.3.2 Flow Type

A syntax for forming URNs for the QoS Flow table's Type parameter is defined for the Session Description Protocol (SDP) as defined by RFC 4566 [14]. Additional standard or vendor-specific URNs can be defined following the standard syntax for forming URNs.

A URN to specify an SDP flow is formed as follows:

```
urn:dslforum-org:sdp-[MediaType]-[Transport]
```

[MediaType] corresponds to the "media" sub-field of the "m" field of an SDP session description.
[Transport] corresponds to the "transport" sub-field of the "m" field of an SDP session description.
Non-alphanumeric characters in either field are removed (e.g., "rtp/avp" becomes "rtpavp").

For example, the following would be valid URNs referring to SDP flows:

```
urn:dslforum-org:sdp-audio-rtpavp
urn:dslforum-org:sdp-video-rtpavp
urn:dslforum-org:sdp-data-udp
```

For flow type URNs following this convention, there is no defined use for TypeParameters, which SHOULD be left empty.

For the ProtocolIdentifier urn:dslforum-org:pppoe, a single flow type is defined referring to the entire PPPoE session. The URL for this flow type is:

```
urn:dslforum-org:pppoe
```

A.3.3 Flow TypeParameters

For the flow type urn:dslforum-org:pppoe, Table 6 specifies the defined TypeParameter values.

Table 6 – Flow TypeParameters values for flow type urn:dslforum-org:pppoe

Name	Description of Value
ServiceName	The PPPoE service name. If specified, only bridged PPPoE sessions designated for the named service would be considered part of this flow. If this parameter is not specified, or is empty, bridged PPPoE associated with any service considered part of this flow.
ACName	The PPPoE access concentrator name. If specified, only bridged PPPoE sessions designated for the named access concentrator would be considered part of this flow. If this parameter is not specified, or is empty, bridged PPPoE associated with any access concentrator considered part of this flow.
PPPDomain	The domain part of the PPP username. If specified, only bridged PPPoE sessions in which the domain portion of the PPP username matches this value are considered part of this flow. If this parameter is not specified, or is empty, all bridged PPPoE sessions are considered part of this flow.

Appendix I: Example RG Queuing Architecture (from TR-059)

The queuing and scheduling discipline envisioned upstream for the RG is shown in Figure 11.

There are multiple access sessions supported in this model, however, all traffic is classified and scheduled in a monolithic system. So, while it might appear at first that the Diffserv queuing and scheduling might apply only to IP-aware access – in fact all access, IP, Ethernet, or PPP is managed by the same system that adheres to the Diffserv model.

For example, at the bottom of the figure, BE treatment is given to the non-IP-aware access sessions (PPPoE started behind the RG or delivered to an L2TP tunnel delivery model). This queue might be repeated several times in order to support fairness among multiple PPPoE accesses – or it can be a monolithic queue with separate rate limiters applied to the various access sessions.

The PTA access is a single block of queues. This is done because NSP access typically works with a single default route to the NSP, and managing more than one simultaneously at the RG would be perilous. The ∑ rate limiter would limit the overall access traffic for a service provider.

Rate limiters are also shown within the EF and AF service classes because the definition of those Diffserv types is based on treating the traffic differently when it falls into various rates.

Finally, at the top of the diagram is the ASP access block of queues. In phase 1A, these queues are provisioned and provide aggregate treatment of traffic mapped to them. In phase 1B, it will become possible to assign AF queues to applications to give them specific treatment instead of aggregate treatment. The EF service class can also require a high degree of coordination among the applications that make use of it so that its maximum value is not exceeded.

Notable in this architecture is that all the outputs of the EF, AF, and BE queues are sent to a scheduler (**S**) that pulls traffic from them in a strict priority fashion. In this configuration EF traffic is, obviously, given highest precedence and BE is given the lowest. The AF service classes fall in-between.

Note that there is significant interest in being able to provide a service arrangement that would allow general Internet access to have priority over other (bulk rate) services.[5] Such an arrangement would be accomplished by assigning the bulk rate service class to BE and by assigning the default service class (Internet access) as AF with little or no committed information rate.

Given this arrangement, the precedence of traffic shown in the figure is arranged as:

1. EF – red dotted line

[5] This "bulk rate" service class would typically be used for background downloads and potentially for peer-to-peer applications as an alternative to blocking them entirely.

2. AF – blue dashed line (with various precedence among AF classes as described in RFC 2597 [10])
3. BE – black solid line

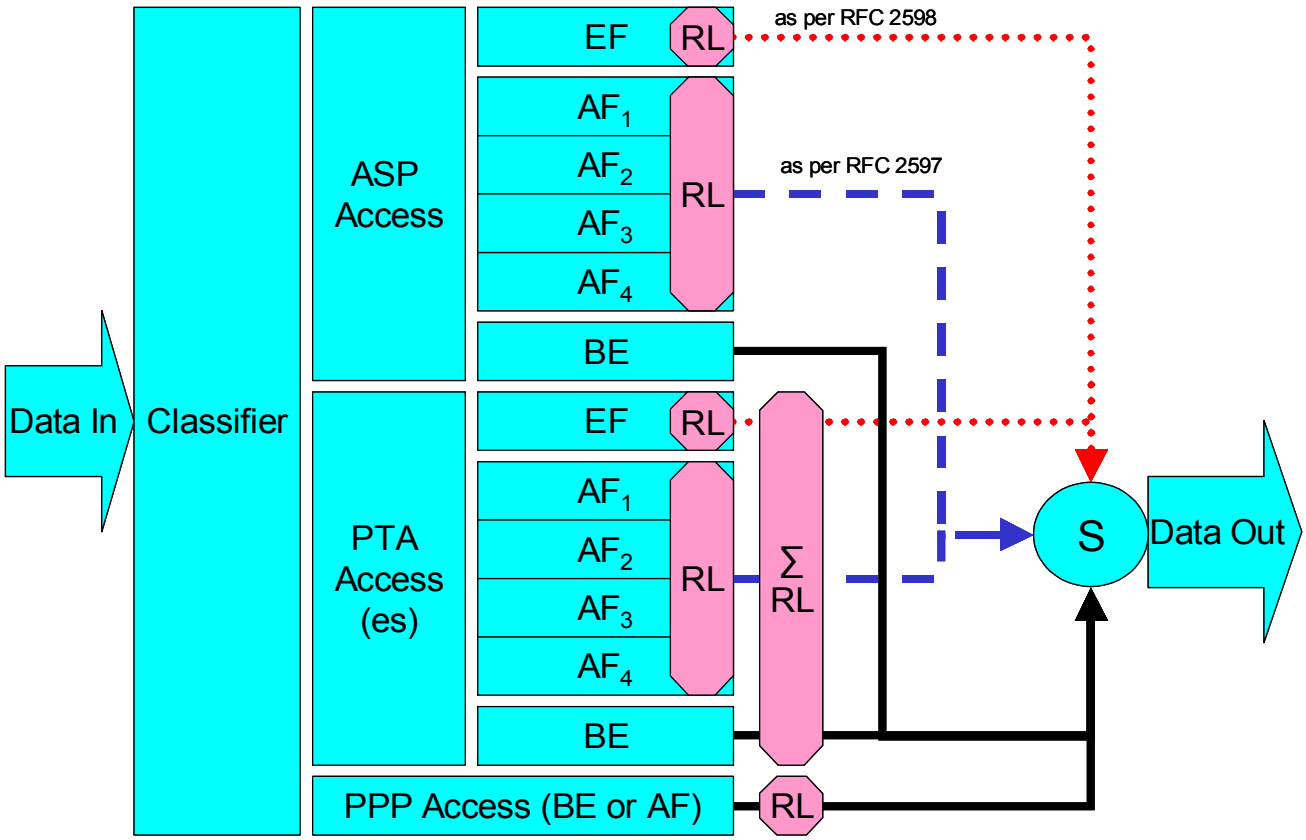

Figure 11 – Queuing and Scheduling Example for RG

In Figure 11 the following abbreviations apply:

ASP – Application Service Provider
PTA – PPP Terminated Aggregation
PPP – Point-to-Point Protocol
EF – Expedited Forwarding – as defined in RFC 3246 [11]
AF – Assured Forwarding – as defined in RFC 2597 [10]
BE – Best Effort forwarding
RL – Rate Limiter
∑RL – Summing Rate Limiter (limits multiple flows)
S – Scheduler

Appendix II: Use of Bridging Objects for VLAN Tagging

In the case of an Ethernet WAN Interface or a VDSL2 WAN Interface based on PTM-EFM, 802.1Q Tagging can be used to tag egress traffic on the WAN interface. This choice enables a multi-VLAN architecture in order to deploy a multi-service configuration (high speed Internet, VoIP, Video Phone, IPTV, etc.), where one VLAN or a group of VLANs are associated with each service. If 802.1Q tagging on the WAN interface is used, it is necessary to have a way to associate LAN incoming 802.1Q tagged or untagged traffic or internally generated traffic (PPPoE, IPoE connections) to the egress (and vice-versa). The solution is to apply coherent bridging rules.

Regarding different traffic bridging rules, the possible cases are characterized as follows:

- Tagged LAN to tagged WAN traffic (pure VLAN bridging), with VLAN ID translation as a special case
- Untagged LAN to tagged WAN traffic
- Internally generated to tagged WAN traffic

To better understand the different cases, refer to Figure 12 and to the following examples.

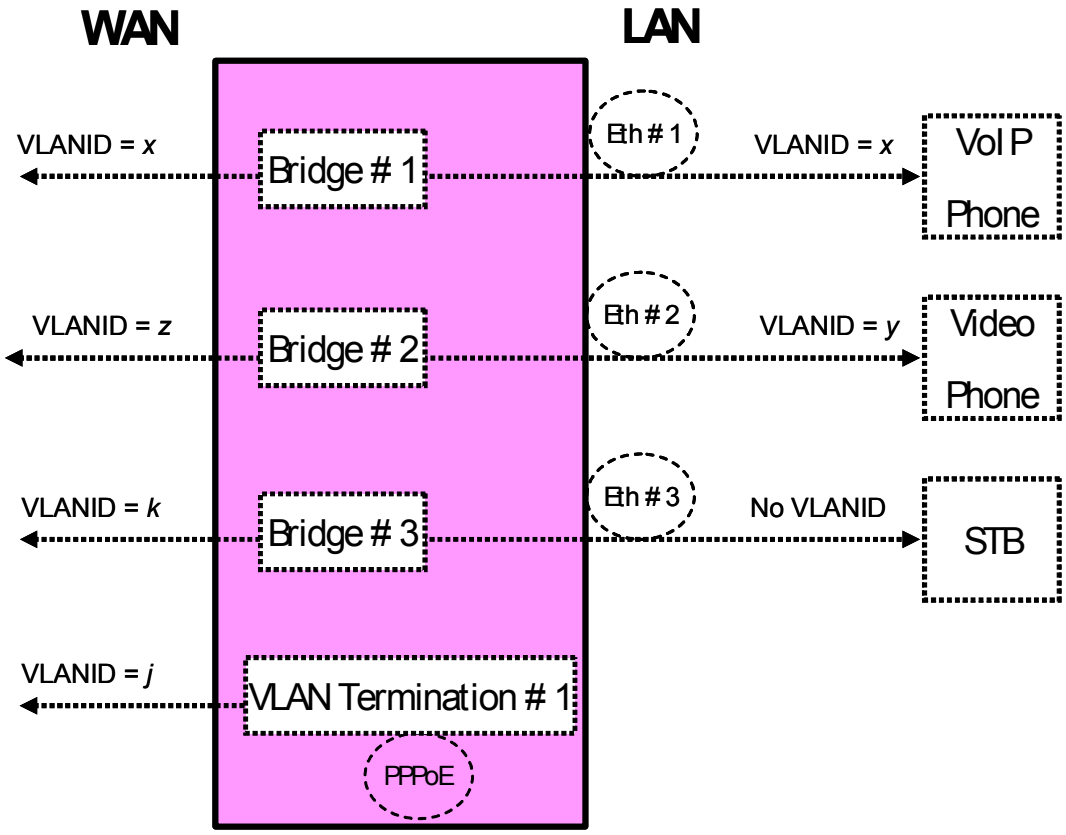

Figure 12 – Examples of VLAN configuration based on Bridging and VLAN Termination objects

II.1 Tagged LAN to Tagged WAN Traffic (VLAN Bridging)

Ethernet port 1 (instance Device.Ethernet.Interface.2) might be dedicated to VoIP service, receiving VLAN ID x tagged traffic from a VoIP phone, and this port would be included in the same bridge dedicated to VoIP service on the WAN interface (instance Device.Ethernet.Interface.1), identified with the same VLAN ID x.

To achieve this, an interface-based bridge would be created using the Bridging object. A Bridge table entry would be created with entries for Ethernet port 1 and the WAN interface and for the VLAN ID x associated with VoIP.

The Bridging model is depicted in Figure 13, while the configuration rules for this situation are summarized in Table 7.

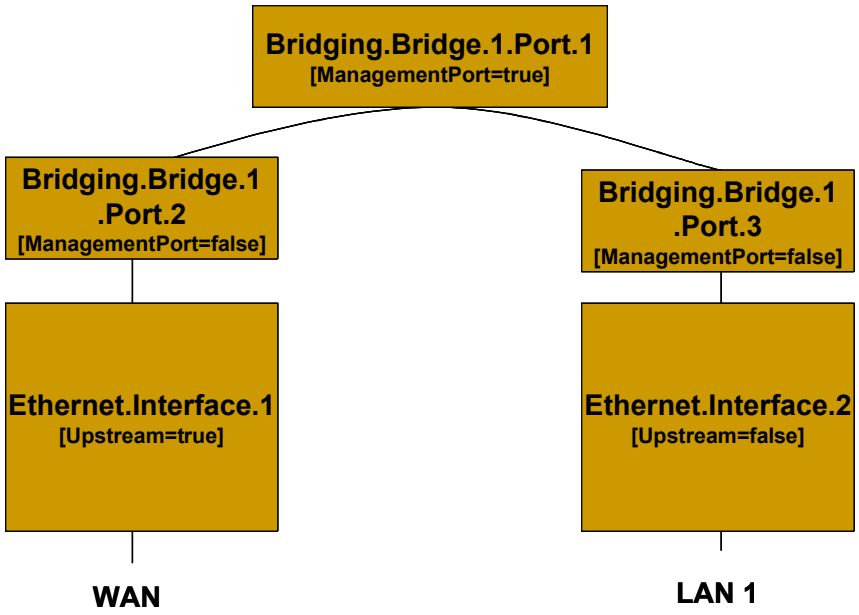

Figure 13 – Bridge 1 model

Table 7 – Tagged LAN to tagged WAN configuration

Description	Bridging TR-069 Configuration
Bridge between WAN and LAN-1 interfaces with VLANID=*x*	[Define VLANx] Device.Bridging.Bridge.1.VLAN.1 : - Name : *VLANx* VLANID : *X*

[Define Ingress Port2-3 – Create an entry for the WAN and LAN port]:

Device.Bridging. Bridge.1.Port.2	-
PVID	*x*
Name	*Port2*
AcceptableFrameTypes	AdmitOnlyVLANTagged

Device.Bridging. Bridge.1.Port.3	-
PVID	*x*
Name	*Port3*
AcceptableFrameTypes	AdmitOnlyVLANTagged

[Associate Egress Port2-3 to VLANx - Create an entry for the WAN and LAN port]

Device.Bridging.Bridge.1.VLANPort.1	-
VLAN	*VLANx*
Port	*Port2*
Untagged	*false*

Device.Bridging.Bridge.1.VLANPort.2	-
VLAN	*VLANx*
Port	*Port3*
Untagged	*false*

II.2 Tagged LAN to Tagged WAN Traffic (Special Case with VLAN ID Translation)

Ethernet port 2 (instance Device.Ethernet.Interface.3) might be dedicated to Video Phone service, receiving VLAN ID y tagged traffic from a Video phone, and this port would be included in the same bridge dedicated to Video Phone service on the WAN interface (instance Device.Ethernet.Interface.1), identified by a different VLAN ID (VLAN ID z). In this case a VLAN translation needs to be performed.

To achieve this, a bridge would be created using the Bridging object. A Bridge table entry would be created along with two associated Filter object entries for {Ethernet port 2/VLAN ID z} and {WAN interface/VLAN ID y}. The Filter identifies the ingress interface and causes the ingress frames to be bridged to the egress VLAN, permitting VLAN-ID translation.

The Bridging model is depicted in Figure 14, while the configuration rules for this situation are summarized in Table 8.

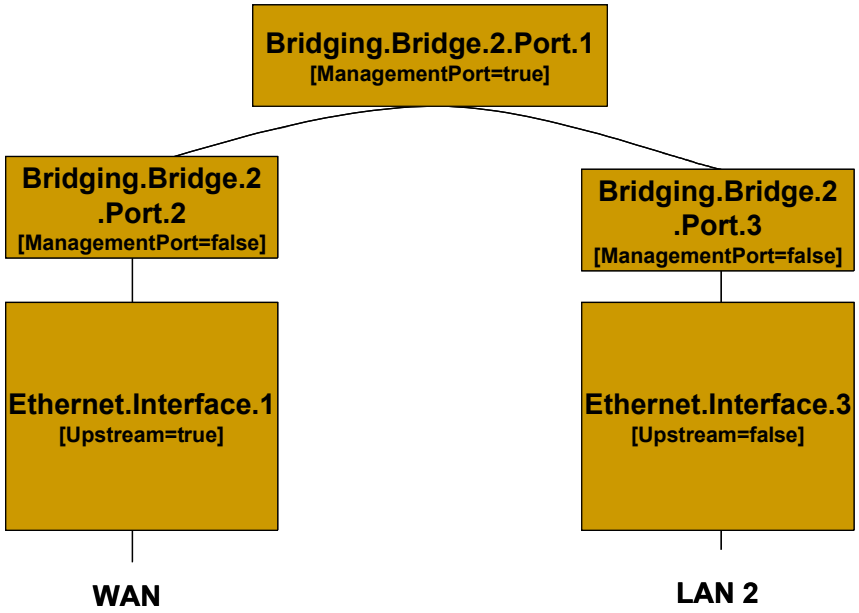

Figure 14 – Bridge 2 model

Table 8 – Tagged LAN to tagged WAN configuration (VLAN ID translation)

Description	Bridging TR-069 Configuration
Tagged LAN Eth-2 to tagged WAN traffic (and vice versa) (special case with VLAN ID translation) WAN VLAN-ID=z LAN VLAN-ID=y	[Define VLANy and VLANz] Device.Bridging.Bridge.2.VLAN.1 | Name | *VLANy* VLANID | *y* Device.Bridging.Bridge.2.VLAN.2 | Name | *VLANz* VLANID | *z*

[Define Ingress Port2 – Create an entry for WAN port]:

Device.Bridging.Bridge.2.Port.2	
PVID	*Z*
Name	*Port2*
AcceptableFrameTypes	AdmitOnlyVLANTagged

[Define Ingress Port3 – Create an entry for the LAN port]:

Device.Bridging.Bridge.2.Port.3	
PVID	*y*
Name	*Port3*
AcceptableFrameTypes	AdmitOnlyVLANTagged

[Associate Egress Port2 to VLANz - Create an entry for WAN port]

Device.Bridging.Bridge.2.VLANPort.1	-
VLAN	*VLANz*
Port	*Port2*
Untagged	*false*

[Associate Egress Port3 to VLANy - Create an entry for each LAN port]

Device.Bridging.Bridge.2.VLANPort.2	-
VLAN	*VLANy*
Port	*Port3*
Untagged	*false*

[Define filter on WAN: ingress from Port 2 is associated with VLANy]

Device.Bridging.Filter.1.	-
Bridge	*VLANy*
Interface	*Port2*

[Define filter on LAN: ingress from Port 3 is associated with VLANz]

Device.Bridging.Filter.2.	-
Bridge	*VLANz*
Interface	*Port3*

II.3 Untagged LAN to Tagged WAN Traffic

Ethernet port 3 (instance Device.Ethernet.Interface.4) might be dedicated to IPTV service, receiving untagged traffic from a STB, and this port would be included in the same bridge dedicated to IPTV service on the WAN interface (instance Device.Ethernet.Interface.1), identified with the VLAN ID k.

To achieve this, an interface-based bridge would be created using the Bridging object. A Bridge table entry would be created, associating in the same bridge untagged frames on Ethernet port 3 with tagged frames on the WAN interface.

The Bridging model is depicted in Figure 15, while the configuration rules for this situation are summarized in Table 9.

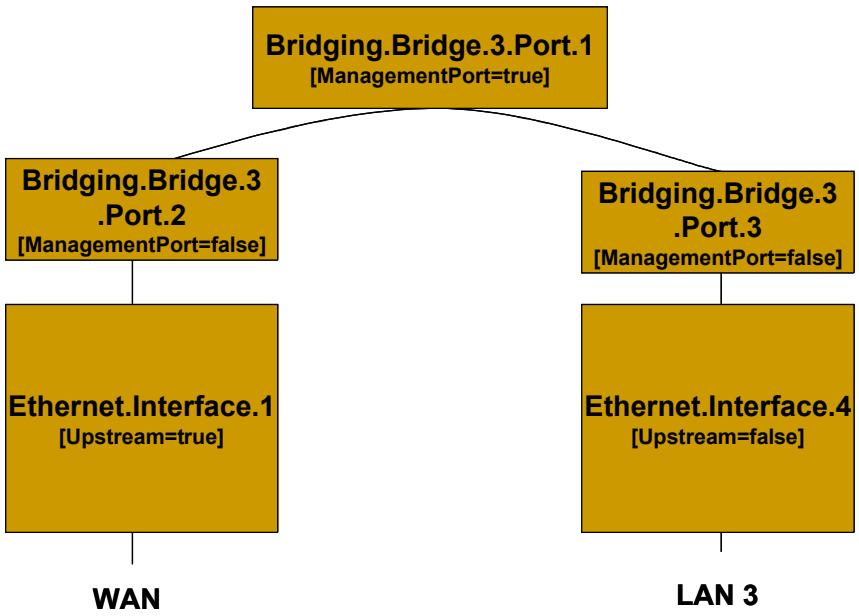

Figure 15 – Bridge 3 model

Table 9 – Untagged LAN to tagged WAN configuration

Description	Bridging TR-069 Configuration
Untagged LAN Eth-3 to tagged WAN (VLAN-ID=k) traffic	[Define VLANk] Device.Bridging.Bridge.3.VLAN.1 | Name | *VLANk* VLANID | *k*

[Define Ingress Port2 – Create an entry for WAN port]:

Device.Bridging.Bridge.3.Port.2	
PVID	*k*
Name	*Port2*
AcceptableFrameTypes	AdmitOnlyVLANTagged

[Define Ingress Port3 – Create an entry for the LAN port]:

Device.Bridging.Bridge.3.Port.3	
Name	*Port3*
AcceptableFrameTypes	AdmitAll

[Associate Egress Port2 to VLANk - Create an entry for WAN port]

Device.Bridging.Bridge.3.VLANPort.1	-
VLAN	*VLANk*
Port	*Port2*
Untagged	*false*

[Associate Egress Port3 to VLANk - Create an entry for each LAN port]

Device.Bridging.Bridge.3.VLANPort.2	-
VLAN	*VLANk*
Port	*Port3*
Untagged	*true*

II.4 Internally Generated to Tagged WAN Traffic

A CPE PPPoE internal session (instance Device.PPP.Interface.1) might be dedicated to Management service and this logical interface would encapsulate/de-encapsulate its outgoing or incoming traffic in the VLAN ID j, dedicated to Management service.

To achieve this, instead of using a bridging object, a VLAN Termination interface would be created (Device.Ethernet.VLANTermination.1). The Bridging model is depicted in Figure 16, while the configuration rules for this situation are summarized in Table 10.

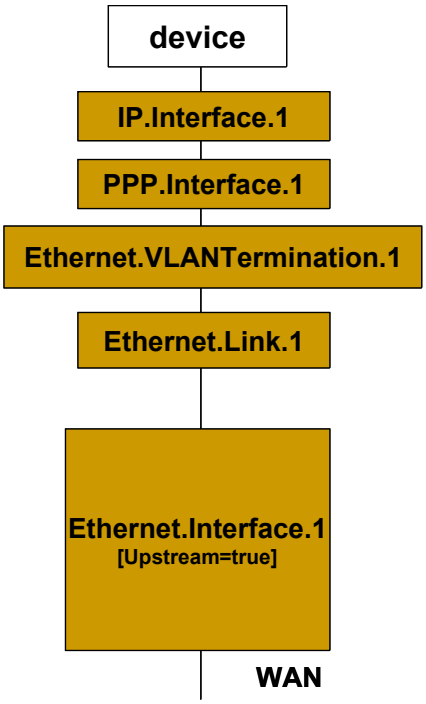

Figure 16 – VLAN Termination model

Table 10 – Internally generated to tagged WAN configuration

Description	VLAN Termination TR-069 Configuration
	[DefineVLAN Termination on top of Ethernet Link]

Device.Ethernet.VLANTermination.1	
VLANID	*j*
LowerLayers	Ethernet.Link.1

II.5 Other Issues

The previous rules can be applied to allow all combinations of traffic. If the subscriber's services are modified, the Bridging configuration might need to be modified accordingly.

It can be interesting to detail the configuration of three special cases:

- More than one LAN interface in a bridge
- 802.1D (re-)marking
- More than one VLAN ID tag for the same LAN interface

II.5.1 More than one LAN Interface in a Bridge

Referring to the example in Section II.1, Tagged LAN to tagged WAN traffic (VLAN bridging), consider adding other Ethernet interfaces (e.g. Ethernet ports 3 and 4 = instance Device. Ethernet.Interface.3/4) to the Video Phone service. The behavior is the same as for the existing Ethernet port 2 (instance Device.Ethernet.Interface.2).

To achieve this, new entries need to be added for interface Eth-3 and Eth-4. The Bridging model is depicted in Figure 17, while the configuration rules for this situation are summarized in Table 7 and Table 11.

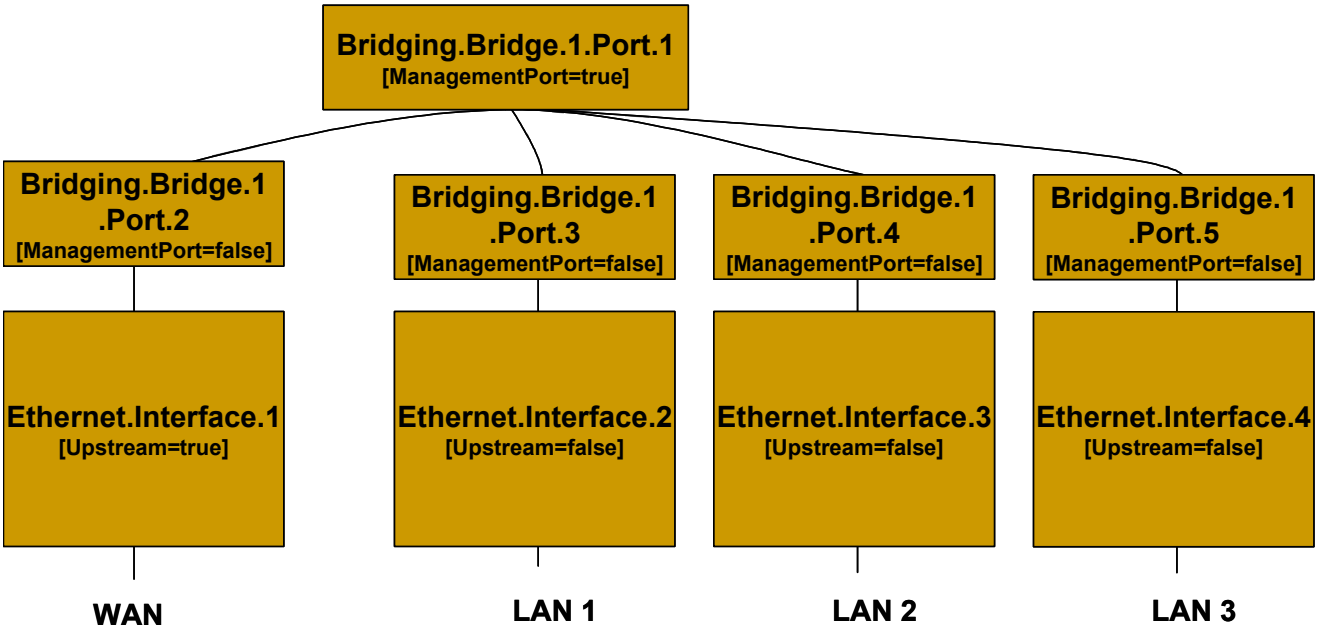

Figure 17 – Bridge 1 model

Table 11 – Configuration to be added to Table 7

<table>
<tr><th>Description</th><th>Bridging TR-069 Configuration</th></tr>
<tr><td>Bridge between WAN and LAN-2/LAN-3 interfaces with VLANID=x

(Configuration to be added to Table 7)</td><td>[Define Ingress Port4-5 – Create an entry for the other LAN ports]:

<table>
<tr><td>Device.Bridging. Bridge.1.Port.4</td><td>-</td></tr>
<tr><td>PVID</td><td>x</td></tr>
<tr><td>Name</td><td>Port4</td></tr>
<tr><td>AcceptableFrameTypes</td><td>AdmitOnlyVLANTagged</td></tr>
</table>

<table>
<tr><td>Device.Bridging. Bridge.1.Port.5</td><td>-</td></tr>
<tr><td>PVID</td><td>x</td></tr>
<tr><td>Name</td><td>Port5</td></tr>
<tr><td>AcceptableFrameTypes</td><td>AdmitOnlyVLANTagged</td></tr>
</table>

[Associate Egress Port4-5 to VLANx - Create an entry for the LAN ports]

<table>
<tr><td>Device.Bridging.Bridge.1.VLANPort.3</td><td>-</td></tr>
<tr><td>VLAN</td><td>VLANx</td></tr>
<tr><td>Port</td><td>Port4</td></tr>
<tr><td>Untagged</td><td>false</td></tr>
</table>

<table>
<tr><td>Device.Bridging.Bridge.1.VLANPort.4</td><td>-</td></tr>
<tr><td>VLAN</td><td>VLANx</td></tr>
<tr><td>Port</td><td>Port5</td></tr>
<tr><td>Untagged</td><td>false</td></tr>
</table>
</td></tr>
</table>

II.5.2 802.1D (Re)-marking

The 802.1Q Tag includes the 802.1D user priority bits field. All the previous cases can also be extended to mark (or re-mark) this 802.1D field. To achieve this, there are different configuration options, one of them is to use the DefaultUserPriority or PriorityRegeneration fields in the Bridge Port object. For untagged frames, more complex rules can be defined referring to the QoS Classification, using the PriorityTagging value. The Bridging configuration rules for marking egress traffic on the WAN interface are summarized in Table 12. Compare it with Table 7.

Table 12 – 802.1D (re-)marking

<table>
<tr><th>Description</th><th>Bridging TR-069 Configuration</th></tr>
<tr><td rowspan="2">802.1D (re-)marking
Remark all WAN egress traffic</td><td>[Mark the ingress frames with Default user Priority, in this case 0]
<table><tr><td>Device.Bridging. Bridge.1. Port.2.</td><td></td></tr><tr><td>DefaultUserPriority</td><td>0</td></tr></table>
[Remark each ingress priority value (0,1,2,3,4,5,6,7) with the priority regeneration string, in this case (0,0,0,0,4,4,4,4)]
<table><tr><td>Device.Bridging. Bridge.1. Port.2.</td><td></td></tr><tr><td>PriorityRegeneration</td><td>0,0,0,0,4,4,4,4</td></tr></table></td></tr>
<tr><td>[In case of ingress untagged frames, for more complex classification, QoS object are referred. In this case remark with 0]
<table><tr><td>Device.Bridging. Bridge.1. Port.2.</td><td></td></tr><tr><td>PriorityTagging</td><td>true</td></tr></table>
<table><tr><td>Device.QoS. Classification. {i}.</td><td></td></tr><tr><td>EthernetPriorityMark</td><td>0</td></tr></table></td></tr>
</table>

II.5.3 More than one VLAN ID Tag Admitted on the Same LAN Interface

Another scenario that can be further detailed is the case of more than one VLAN ID tag admitted on the same LAN interface. A practical example would be a 2 box scenario, with a User Device generating traffic segregated in multiple VLANs (e.g. a router offering services to the customer), and an Internet Gateway Device, providing WAN connectivity to the Access Network, with the connection between the two pieces of equipment using an Ethernet interface.

In this case, we assume the User Device is able to tag the different traffic flows, segregating the different services (Voice, Video, …) into different VLANs. The IGD needs, on the same LAN interface, to be able to receive different VLAN ID and correctly forward or translate to the WAN interface (and vice versa). To achieve this, appropriate Bridging objects need to be configured.

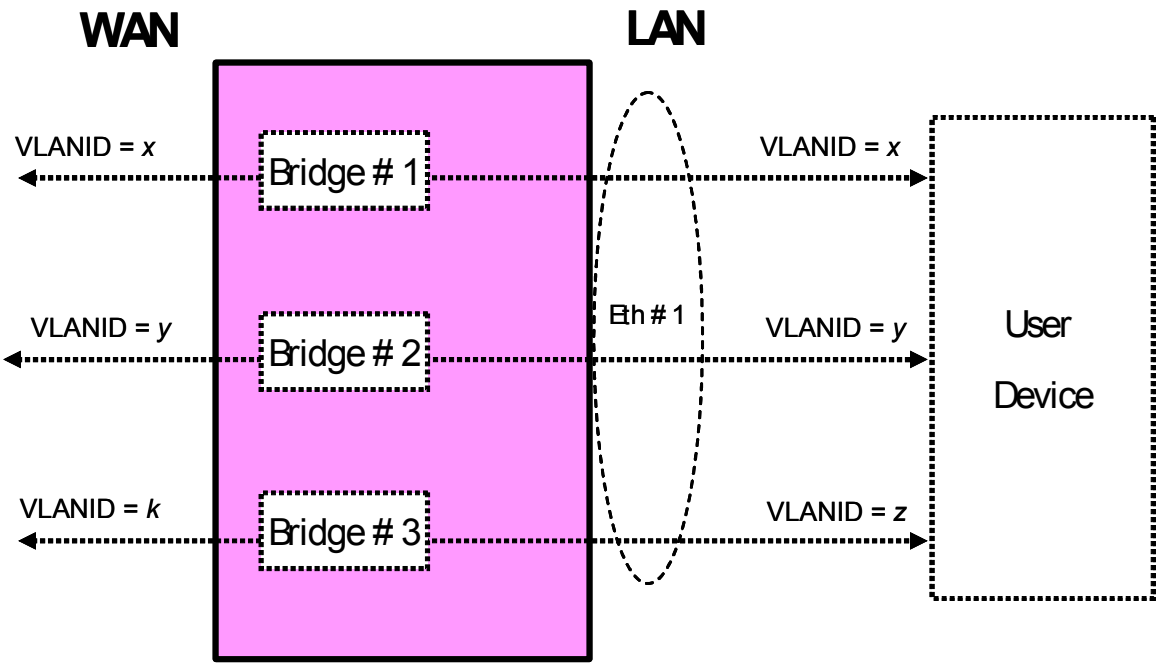

Figure 18 – Example of VLAN configuration in a 2 box scenario

Referring to Figure 18 as an example, assume the case of three VLANs (VLAN ID=x,y,z) offered by a User Device to the IGD on the same LAN interface (Eth-1). The IGD bridges two of them (VLAN ID=x,y) and translates the other one (VLAN ID=z) to the WAN interface (VLAN ID=k).

On the IGD, this can be achieved using a combination of the Bridging objects detailed in the preceding sections, with 3 bridge entries and their related entries. Refer to Figure 19 for the Bridging model and Table 13 for the global configuration.

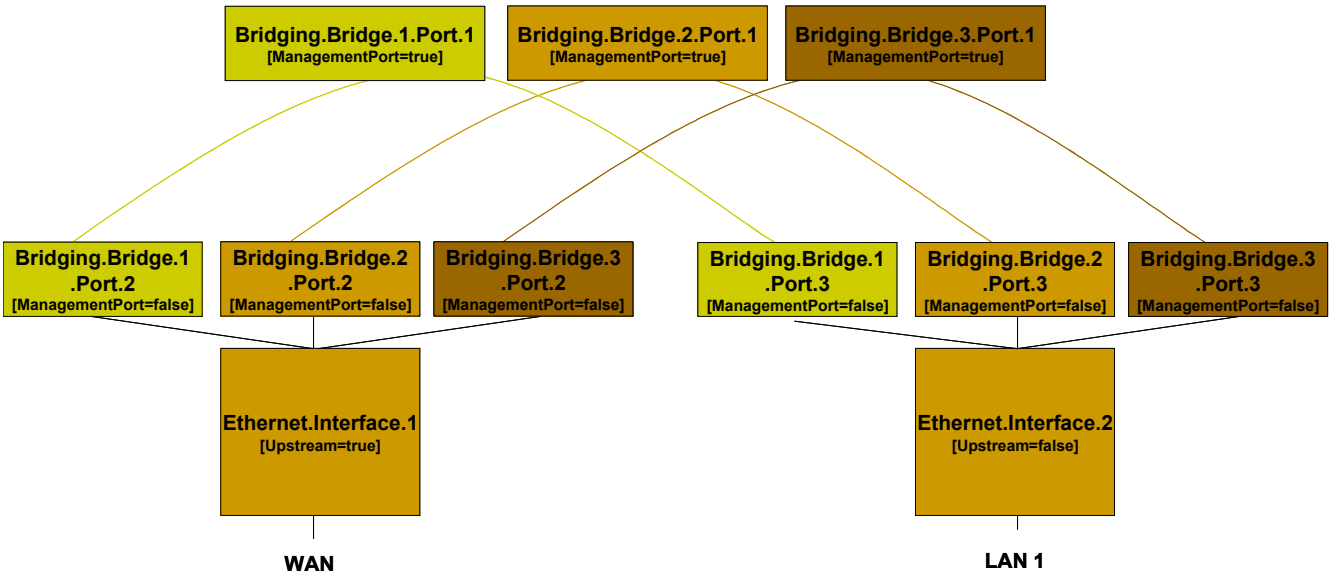

Figure 19 – Bridge 1,2,3 model

Table 13 – More than one VLAN ID tag admitted on the same LAN interface

<table>
<tr><th>Description</th><th>Bridging TR-069 Configuration</th></tr>
<tr><td>More than one VLAN ID tag admitted on the same LAN interface</td><td>The configuration is the sum of Sections II.1 and II.2, but on the LAN side the lower layer to be configured for each Bridge Port is always: Ethernet.Interface.2

<table>
<tr><td>Device.Bridging. Bridge.1. Port.3.</td><td></td></tr>
<tr><td>LowerLayers</td><td>Ethernet.Interface.2</td></tr>
</table>
<table>
<tr><td>Device.Bridging. Bridge.2. Port.3.</td><td></td></tr>
<tr><td>LowerLayers</td><td>Ethernet.Interface.2</td></tr>
</table>
<table>
<tr><td>Device.Bridging. Bridge.3. Port.3.</td><td></td></tr>
<tr><td>LowerLayers</td><td>Ethernet.Interface.2</td></tr>
</table>
</td></tr>
</table>

Appendix III: Theory of Operations

This section discusses the theory of operations for various technologies found within the Device:2 data model.

III.1 Wi-Fi

III.1.1 Multi-radio and Multi-band Wi-Fi Radio Devices

The WiFi.Radio object description says "This object models an 802.11 wireless radio on a device. If the device can establish more than one connection simultaneously (e.g. a dual radio device), a separate WiFi.Radio instance MUST be used for each physical radio of the device."

The following sections clarify when multiple WiFi.Radio instances are needed, and the impact on their underlying parameters in the case of multi-radio and/or multi-band devices.

III.1.2 Definitions

Each physical radio allows the transmission and reception of data on a single Wi-Fi channel at a given time. A single-radio device is able to transmit/receive a packet at a given time only on one Wi-Fi channel. Similarly, a dual-radio device is able to simultaneously transmit/receive data on two Wi-Fi channels. In general, a device with N radios is able to simultaneously transmit/receive data on N Wi-Fi channels.

An important point is that Wi-Fi can operate at two different frequency bands, 2.4 GHz and 5 GHz, as follows:

- Wi-Fi technologies based on IEEE 802.11b/g standard operate on the 2.4 GHz frequency band.
- Wi-Fi technologies based on IEEE 802.11a standard operate on the 5 GHz frequency band.
- Wi-Fi technologies based on IEEE 802.11n standard operate on both the 2.4 and 5 GHz frequency bands.

Radios that operate at a single frequency band (e.g. 2.4 GHz only 802.11b/g devices) are called single-band radios. Radios that can operate at both 2.4 and 5 GHz frequency bands (e.g. 802.11a/b/g devices) are called dual-band radios.

A dual-band device can be a single-radio device if it can be configured to operate at 2.4 or 5 GHz frequency bands. However, only a single frequency band is used to transmit/receive at a given time. In such a case the device has a single physical radio that is dual-band.

Also, a dual-radio single-band device can exist (although uncommon) if both radios are single-band.

III.1.3 Number of Instances of WiFi.Radio Object

Given the definitions above, a separate WiFi.Radio instance will be used for each physical radio of the device, i.e. one instance for a single-radio device, two instances for dual-radio devices, and so on. A single WiFi.Radio instance will therefore be used for a dual-band single-radio device.

Each WiFi.Radio instance is configured separately and is, in general, completely independent of other instances.

III.1.4 SupportedFrequencyBands and OperatingFrequencyBand

The frequency band used by a WiFi device is an important parameter. With first generations of WiFi technologies, the specific frequency band was linked to the IEEE standard in use (i.e. 802.11b/g are 2.4 GHz standards, while 802.11a is a 5 GHz standard). With the introduction of the IEEE 802.11n standard, which can work both at 2.4 and 5 GHz, two specific parameters are used to indicate the supported frequency bands and the operating frequency band.

SupportedFrequencyBands is a list-valued parameter, containing one item for single-band radios (either 2.4GHz or 5GHz) and two items for dual-band radios (both 2.4GHz and 5GHz).

The OperatingFrequencyBand parameter specifies which frequency band is currently being used by a dual-band radio (i.e. set to one of the two items listed in the SupportedFrequencyBands parameter). For single-band radios, OperatingFrequencyBand always has the same value as SupportedFrequencyBands (since only one frequency band is supported).

III.1.5 Behavior of Dual-band Radios when OperatingFrequencyband Changed

When the configured operating frequency band of a dual-band radio is changed (i.e. the value of the OperatingFrequencyBand parameter is modified), this has an impact on other parameters within the WiFi.Radio object.

The Channel parameter has to be changed, since channels for the 2.4 GHz frequency band are in the range 1-14, while channels for the 5 GHz frequency band are in the range 36-165 (at least in the USA and Europe). The expected behavior is that, upon modifying the OperatingFrequencyBand parameter, the device automatically selects a new channel that is valid for the new frequency band (according to some vendor-specific selection procedure).

Persistence of the Channel parameter value for the previous frequency band is not required. For example, if OperatingFrequencyBand is later changed back to *5GHz*, a new valid value for the Channel parameter is automatically selected by the device, but this value need not be the same as was selected the last time OperatingFrequencyBand was set to *5GHz*.

Other parameters whose values can be impacted when the OperatingFrequencyBand changes, include: ExtensionChannel, PossibleChannels, SupportedStandards, OperatingStandards, IEEE80211hSupported, and IEEE80211hEnabled. If the current value is no longer valid, the device will automatically select a valid new value according to some vendor-specific procedure, and the old value need not persist.

III.1.6 SupportedStandards and OperatingStandards

The SupportedStandards parameter is a list of all IEEE 802.11 physical layer modes supported by the devices. Wi-Fi is in general backward compatible, so 802.11g devices are also 802.11b devices, 802.11n devices are also 802.11b/g devices (if operating at 2.4 GHz), and 802.11n devices are also 802.11a devices (if operating at 5 GHz).

For dual-band radios, the OperatingFrequencyBand parameter is used for switching the operating frequency band. For this reason SupportedStandards only includes those values corresponding to operation in the frequency band indicated by the OperatingFrequencyBand parameter. For example, for dual-band 802.11a/b/g/n devices, SupportedStandards can be *b, g, n* when OperatingFrequencyBand is *2.4GHz* and *a, n* when OperatingFrequencyBand is *5GHz*.

The OperatingStandards parameter is used to limit operation to a subset of physical modes supported. For example, a 802.11b/g/n radio will have *b, g, n* value for the SupportedStandards parameter, but can be configured to operate only with 802.11n by setting the OperatingStandards parameter to *n*.

Appendix IV: Use Cases

This section presents a number of management-related use cases that correspond to typical ACS activities.

IV.1 Create a WAN Connection

The ACS can create the objects in the interface stack bottom-up. Each time a new higher-layer object is created, the link with the underlying interface object needs to be set. The layer 1 interface, in this case a DSL.Channel and DSL.Line object, will already exist (ACS can not create physical interfaces).

1. The ACS uses AddObject to create a new ATM.Link object, a new Ethernet.Link object, a new PPP.Interface object, and a new IP.Interface object.
2. The LowerLayers parameter in an existing DSL.Channel object is already linked to an existing DSL.Line object (ACS can not configure this linkage).
3. The ACS uses SetParameterValues to configure the new objects including enabling the objects and using the LowerLayers parameters as follows:
    a. Setting the LowerLayers parameter in the ATM.Link object to link it to an existing DSL.Channel object that is configured with ATM encapsulation (i.e. the read-only LinkEncapsulationUsed parameter in the DSL.Channel object is set to one of the ATM-related enumeration values).
    b. Setting the LowerLayers parameter in the Ethernet.Link object to link it to the ATM.Link object.
    c. Setting the LowerLayers parameter in PPP.Interface object to link it to the Ethernet.Link object.
    d. Setting the LowerLayers parameter in IP.Interface object to link it to the PPP.Interface object.
4. The CPE updates the InterfaceStack table automatically. The stack looks like this: IP.Interface → PPP.Interface → Ethernet.Link → ATM.Link → DSL.Channel → DSL.Line.
5. Note that the ACS might also want to update other related objects, including the NAT object, the Routing.Router object, or various QoS and Bridging tables. VLANs might also need to be created.

IV.2 Modify a WAN Connection

In this use case the ACS needs to modify an existing WAN connection, in order to insert a new layer in the stack or to change some portion of the interface stack. This is not the management WAN connection. For the purposes of this example, the ACS is changing the WAN connection in use case IV.1 to make use of PTM rather than ATM-based aggregation.

1. The ACS uses AddObject to create a new PTM.Link object.
2. The ACS uses SetParameterValues to configure the objects, including enabling the new PTM.Link object and using the LowerLayers parameter as follows:
    a. Setting the LowerLayers parameter in the PTM.Link object to link it to an existing DSL.Channel object that is configured with PTM encapsulation (i.e. the read-only LinkEncapsulationUsed parameter in the DSL.Channel object is set to one of the PTM-related enumeration values).

   b. Setting the LowerLayers parameter in the Ethernet.Link object to refer to the PTM.Link object rather than the ATM.Link object.
   c. Setting the LowerLayers parameter in the IP.Interface object to refer to the Ethernet.Link object rather than the PPP.Interface object.
3. The CPE updates the InterfaceStack table automatically. The stack looks like this: IP.Interface → Ethernet.Link → PTM.Link → DSL.Channel → DSL.Line.
4. Note that the ACS might also want to update other related objects, including the Bridging table. The ACS might also want to delete the existing PPP.Interface and ATM.Link objects.

IV.3 Delete a WAN Connection

Assume that we want to delete the WAN connection as it is configured in use case IV.1.

1. The ACS uses DeleteObject to delete the IP.Interface object.
2. The ACS uses DeleteObject to delete the PPP.Interface object.
3. The ACS uses DeleteObject to delete the Ethernet.Link object.
4. As each of these objects is deleted, the InterfaceStack is adjusted automatically by the CPE.
5. Any strong references to the deleted objects, e.g. in Device.QoS classification rules, will automatically be set to empty strings.

IV.4 Discover whether the Device is a Gateway

Many operators want to determine if a particular device is a "gateway" or not. The term "gateway", however, is rather vague; usually the operator wants to know one (or more) of the following things:

1. If the device terminates the WAN connection(s).
2. If the device is responsible for providing DHCP addresses to the other devices in the home.
3. If the device provides functionality such as NAT or routing capabilities.

In order to determine if the device terminates a WAN connection, the ACS might look for an interface object with a technology that is by definition WAN (such as DSL) or for a technology that could be a WAN termination technology (such as Ethernet or MoCA).

In order to determine if the device is responsible for providing addresses to other devices in the home, the ACS could check for the existence of the DHCP Server object. The existence of the Host table also indicates that the device is aware of Hosts, by whatever means they're addressed. The existence of the ManageableDevice table within the ManagementServer object also indicates that the device serves as the DHCP server for the TR-069 managed device exchange defined in TR-069 [2] Annex G, which is also often an indication of "gateway" functionality.

In order to determine if the device provides functionality such as NAT or a router, the ACS would check for the existence of an enabled NAT or Routing.Router object.

IV.5 Provide Extended Home Networking Topology View

Another use case is to determine the topology of the home network behind the gateway. For a generic understanding of the network, the Host table provides information such as the layer 2 and layer 3 interfaces via which the Host is connected as well as DHCP lease information for each connected Host.

If the operator is interested in UPnP devices in the home network, the UPnP.Discovery tables (RootDevice, Device, and Service) provide that information in addition to the Host table entries that correspond to a particular UPnP Root Device, Device, or Service. Finally, the ManageableDevice table provides information about the TR-069 managed devices that the CPE has learned about through the DHCP message exchange defined in TR-069 [2] Annex G.

IV.6 Determine Current Interfaces Configuration

One of the most fundamental ACS tasks is to determine the general picture of the interfaces for a device so that it can understand which WAN and LAN side connections exist. In the InternetGatewayDevice:1 data model, for example, the ACS would use the GetParameterNames and/or GetParameterValues RPCs to find the available WANDevice, WANConnectionDevice, and WAN**Connection instances, with hierarchical containment implying interface layers. In the Device:2 data model, it would work this way:

1. The ACS would issue a GetParameterValues for the InterfaceStack table. This table would provide a list of all the Interface connections. The ACS could use this table to build up the general picture of the Interfaces that are part of the current configuration.
2. If the ACS is interested in the specifics of an individual interface, it can then go and issue GetParameterNames or GetParameterValues for the interfaces of interest.

IV.7 Create a WLAN Connection

In this use case the ACS creates a new WLAN connection. For the purposes of illustration, in this example the ACS will create a new SSID object to link to an existing radio (a new SSID object implies a different SSID value than those used by existing WiFi connections). The layer 1 interface, in this case a WiFi.Radio object, will already exist (ACS can not create physical interfaces).

1. The ACS uses AddObject to create a new WiFi.SSID object and WiFi.AccessPoint object.
2. The ACS uses SetParameterValues to configure the new WiFi.SSID object, including enabling it and setting the value of the LowerLayers parameter to reference the device's WiFi.Radio object.
3. The ACS uses SetParameterValues to add the new WiFi.SSID object to the LowerLayers parameter of an existing non-management Bridging.Bridge.{i}.Port object, as appropriate. Note: a non-management bridge port is indicated when its ManagementPort parameter is set to false.
4. The ACS uses SetParameterValues to configure the new WiFi.AccessPoint object, including enabling it and setting the value of its SSIDReference parameter to reference the WiFi.SSID object.
5. The CPE updates the InterfaceStack table automatically.

6. Note that the ACS might also want to update other related objects; also, if there were no appropriate existing bridge port to which to connect the SSID, the ACS might need to create that object as well.

IV.8 Delete a WLAN Connection

In this use case the ACS deletes the SSID created in use case IV.7.

1. The ACS uses DeleteObject to delete the WiFi.SSID object and WiFi.AccessPoint object.
2. The CPE automatically updates the InterfaceStack table.
3. Note that if the radio has no other SSIDs configured, this would operationally disable the wireless interface.

IV.9 Configure a DHCP Client and Server

In this use case, the ACS wants to configure a DHCP server to provide private 192.168.1.x IP addresses to most home network (HN) devices, but to obtain IP addresses from the network for HN devices that present an option 60 (vendor class ID) value that begins with "ACME".

The ACME devices are remotely managed, so the ACS will also configure the DHCP clients on those devices and the DHCP server on the gateway.

IV.9.1 DHCP Client Configuration (ACME devices)

The ACME devices are quite simple. Each has a single wired Ethernet port and a single IP interface.

A DHCP Client object is created and configured as follows:

DHCPv4.Client.1.Enable	*true*
DHCPv4.Client.1.Interface	Device.IP.Interface.1
DHCPv4.Client.1.SentOption.1.Enable	*true*
DHCPv4.Client.1.SentOption.1.Tag	60
DHCPv4.Client.1.SentOption.1.Value	"ACME Widget" (as hexBinary)

IV.9.2 DHCP Server Configuration (gateway)

The gateway is also relatively simple. Its LAN IP interface is IP.Interface.1.

A DHCP Server object is created and configured as follows:

DHCPv4.Server.Enable	*true*
DHCPv4.Relay.Enable	*true*
DHCPv4.Relay.Forwarding.1.Enable	*true*
DHCPv4.Relay.Forwarding.1.Interface	Device.IP.Interface.1

DHCPv4.Relay.Forwarding.1.VendorClassID	"ACME"
DHCPv4.Relay.Forwarding.1.VendorClassIDMode	"Prefix"
DHCPv4.Relay.Forwarding.1.LocallyServed	*false*
DHCPv4.Relay.Forwarding.1.DHCPServerIPAddress	1.2.3.4
DHCPv4.Server.Pool.1.Enable	*true*
DHCPv4.Server.Pool.1.Interface	Device.IP.Interface.1
DHCPv4.Server.Pool.1.MinAddress	192.168.1.64
DHCPv4.Server.Pool.1.MaxAddress	192.168.1.254
DHCPv4.Server.Pool.1.ReservedAddresses	192.168.1.128, 192.168.1.129
DHCPv4.Server.Pool.1.SubnetMask	255.255.255.0

If a DHCP request includes an option 60 value that begins with "ACME", the request is forwarded to the DHCP server at 1.2.3.4. All other requests are served locally from the pool 192.168.1.64 - 192.168.1.254 (excluding 192.168.1.128 and 192.168.1.129).

IV.10 Reconfigure an Existing Interface

The ACS might want to reconfigure an existing Interface to provide alternate routing functionality. For the purposes of this illustration, an existing Ethernet Interface that is configured for the LAN-side will be reconfigured as a WAN-side Ethernet Interface replacing an existing DSL Interface.

The current configuration on the WAN side looks like:
IP.Interface.1 → Ethernet.Link.1 → ATM.Link.1 → DSL.Channel.1 → DSL.Line.1

The current configuration on the LAN side contains:

- IP.Interface.2 → Ethernet.Link.2 → Bridging.Bridge.1.Port.1 (ManagementPort=true)
- Bridging.Bridge.1.Port.1 LowerLayers parameter has two references:
  - Bridging.Bridge.1.Port.2
  - Bridging.Bridge.1.Port.3
- Bridging.Bridge.1.Port.2 LowerLayers parameter has a reference of Ethernet.Interface.1
- Bridging.Bridge.1.Port.3 LowerLayers parameter has a reference of Ethernet.Interface.2

The ACS would follow these steps to reconfigure the Ethernet.Interface:

1. Determine which Ethernet.Interface is to be reconfigured. For the purpose of this illustration we will use Ethernet.Interface.1.
2. Use GetParameterValues to retrieve the InterfaceStack.
3. Find the higher-layer Interface of Ethernet.Interface.1 by finding the InterfaceStack entry that has Ethernet.Interface.1 as the LowerLayer. The HigherLayer parameter of the identified InterfaceStack instance will be the Interface we are interested in, for the purpose of this illustration we found Bridging.Bridge.1.Port.2.

4. Use DeleteObject to remove Bridging.Bridge.1.Port.2. This removal will automatically clean up the InterfaceStack instances that connect Bridging.Bridge.1.Port.1 → Bridging.Bridge.1.Port.2 and Bridging.Bridge.1.Port.2 → Ethernet.Interface.1. Also, it will remove Bridging.Bridge.1.Port.2 from the LowerLayers parameter contained within Bridging.Bridge.1.Port.1.
5. Find the DSL.Line reference within the LowerLayer parameter of the InterfaceStack.
6. Follow the InterfaceStack up to the Ethernet.Link reference by looking at the HigherLayer parameter in the current InterfaceStack instance and then finding the InterfaceStack instance containing that Interface within the LowerLayer parameter until the HigherLayer reference is the Ethernet.Link Interface. For the purpose of this illustration, we found Ethernet.Link.1.
7. Use SetParameterValues to reconfigure the LowerLayers parameter of Ethernet.Link.1 such that its value is "Device.Ethernet.Interface.1" instead of "Device.ATM.Link.1".
8. The CPE updates the InterfaceStack table and sets the Upstream parameter to true on the Ethernet.Interface.1 instance automatically.
9. Note that the ACS might also want to update other related objects, including the NAT object, the Routing.Router object, or various QoS and Bridging tables. VLANs might also need to be created.

After the CWMP Session is completed and the CPE commits the configuration, the WAN side will look like:
IP.Interface.1 → Ethernet.Link.1 → Ethernet.Interface.1

End of Broadband Forum Technical Report TR-181

