BasicManagement:2

Se ce Te a e Ve 1.01

F UP P Ve 1.0
S a : *Standardized DCP (SDCP)*
Da e: *February 16th, 2012*

This Standardized DCP has been adopted as a Standardized DCP by the Steering Committee of the UPnP Forum, pursuant to Section 2.1(c)(ii) of the UPnP Forum Membership Agreement. UPnP Forum Members have rights and licenses defined by Section 3 of the UPnP Forum Membership Agreement to use and reproduce the Standardized DCP in UPnP Compliant Devices. All such use is subject to all of the provisions of the UPnP Forum Membership Agreement.

A h	C a
William Lupton (Editor)	2Wire / Pace
Davide Moreo	Telecom Italia
Francois-Gaël Ottogalli	France Telecom Group
Xavier Roubaud	France Telecom Group
Wouter van der Beek	Philips
Kiran Vedula	Samsung Electronics

The UPnP Forum in no way guarantees the accuracy or completeness of this author list and in no way implies any rights for or support from those members listed. This list is not the specifications' contributor list that is kept on the UPnP Forum's website.

C e

1.1. INTRODUCTION 7
1.2. REFERENCES 7
1.3. GLOSSARY 9
1.4. NOTATION 9
1.4.1. Data Types *10*
1.4.2. Strings Embedded in Other Strings *10*
1.5. DERIVED DATA TYPES 11
1.5.1. Comma Separated Value (CSV) Lists *11*
1.5.2. Embedded XML Documents *13*
1.6. MANAGEMENT OF XML NAMESPACES IN STANDARDIZED DCPS 13
1.6.1. Namespace Names, Namespace Versioning and Schema Versioning *15*
1.6.2. Namespace Usage Examples *17*
1.7. VENDOR-DEFINED EXTENSIONS 17

2.1. SERVICETYPE 18
2.2. KEY CONCEPTS 18
2.2.1. Maintenance *18*
2.2.2. Diagnostic tests *18*
2.2.3. Logging *20*
2.2.4. Security *20*
2.3. STATE VARIABLES 22
2.3.1. DeviceStatus *25*
2.3.2. SequenceMode *25*
2.3.3. TestIDs *26*
2.3.4. ActiveTestIDs *26*
2.3.5. LogURIs *27*
2.3.6. A_ARG_TYPE_Boolean *27*
2.3.7. A_ARG_TYPE_String *27*
2.3.8. A_ARG_TYPE_UShort *27*
2.3.9. A_ARG_TYPE_UInt *27*
2.3.10. A_ARG_TYPE_DateTime *27*
2.3.11. A_ARG_TYPE_MSecs *27*
2.3.12. A_ARG_TYPE_RebootStatus *27*
2.3.13. A_ARG_TYPE_TestID *28*
2.3.14. A_ARG_TYPE_TestType *28*
2.3.15. A_ARG_TYPE_TestState *28*
2.3.16. A_ARG_TYPE_TestEndpoint *29*
2.3.17. A_ARG_TYPE_TestSchedule *29*
2.3.18. A_ARG_TYPE_TestSessID *29*
2.3.19. A_ARG_TYPE_DSCP *29*
2.3.20. A_ARG_TYPE_Host *29*
2.3.21. A_ARG_TYPE_Hosts *30*
2.3.22. A_ARG_TYPE_HostName *30*
2.3.23. A_ARG_TYPE_PingStatus *30*
2.3.24. A_ARG_TYPE_NSLookupStatus *30*
2.3.25. A_ARG_TYPE_NSLookupResult *30*
2.3.26. A_ARG_TYPE_TracerouteStatus *32*
2.3.27. A_ARG_TYPE_BandwidthTestInfo *32*
2.3.28. A_ARG_TYPE_BandwidthTestSpec *37*
2.3.29. A_ARG_TYPE_BandwidthTestStatus *40*
2.3.30. A_ARG_TYPE_BandwidthTestResult *40*

2.3.31. A_ARG_TYPE_Interfaces 43
2.3.32. A_ARG_TYPE_InterfaceResetStatus 44
2.3.33. A_ARG_TYPE_LogURI 44
2.3.34. A_ARG_TYPE_LogURL 44
2.3.35. A_ARG_TYPE_LogLevel 45
2.3.36. A_ARG_TYPE_LogMaxSize 45
2.3.37. A_ARG_TYPE_ACL 45
2.4. EVENTING AND MODERATION 47
2.4.1. SSDP Announcement Mechanism 48
2.5. ACTIONS 49
2.5.1. Reboot() 50
2.5.2. BaselineReset() 52
2.5.3. GetDeviceStatus() 53
2.5.4. SetSequenceMode() 54
2.5.5. GetSequenceMode() 55
2.5.6. Ping() 55
2.5.7. GetPingResult() 57
2.5.8. NSLookup() 58
2.5.9. GetNSLookupResult() 59
2.5.10. Traceroute() 60
2.5.11. GetTracerouteResult() 61
2.5.12. GetBandwidthTestInfo() 62
2.5.13. BandwidthTest() 63
2.5.14. GetBandwidthTestResult() 64
2.5.15. InterfaceReset() 65
2.5.16. GetInterfaceResetResult() 67
2.5.17. SelfTest() 68
2.5.18. GetSelfTestResult() 69
2.5.19. GetTestIDs() 70
2.5.20. GetActiveTestIDs() 70
2.5.21. GetTestInfo() 71
2.5.22. CancelTest() 72
2.5.23. GetLogURIs() 73
2.5.24. SetLogInfo() 73
2.5.25. GetLogInfo() 74
2.5.26. GetACLData() 75
2.5.27. Common Error Codes 76
2.6. BANDWIDTH TESTS (NORMATIVE) 78
2.6.1. Approach 78
2.6.2. Protocol 81
2.6.3. Capabilities 82
2.6.4. Settings 83
2.6.5. Results 85
2.6.6. Profiles 87
2.6.7. Extensibility 92
2.7. THEORY OF OPERATION (INFORMATIVE) 93
2.7.1. Assumptions 93
2.7.2. Rebooting the Parent Device 94
2.7.3. Resetting the Parent Device 96
2.7.4. Using Sequence Mode 97
2.7.5. Running a Ping Test 100
2.7.6. Running an NSLookup Test 100
2.7.7. Running a Traceroute Test 101
2.7.8. Running an InterfaceReset Test 102
2.7.9. Running a Self Test 103
2.7.10. Bandwidth Tests 104

2.7.11. Manipulating Logs 114

L f Tab e

Table 1-1: CSV Examples 12

Table 1-2: Namespace Definitions 14

Table 1-3: Schema-related Information 14

Table 2-1: State Variables 22

Table 2-2: allowedValueList for *A_ARG_TYPE_RebootStatus* 27

Table 2-3: allowedValueList for *A_ARG_TYPE_TestType* 28

Table 2-4: allowedValueList for *A_ARG_TYPE_TestState* 28

Table 2-5: allowedValueList for *A_ARG_TYPE_TestEndpoint* 29

Table 2-6: allowedValueList for *A_ARG_TYPE_PingStatus* 30

Table 2-7: allowedValueList for *A_ARG_TYPE_NSLookupStatus* 30

Table 2-8: allowedValueList for *A_ARG_TYPE_TracerouteStatus* 32

Table 2-9: allowedValueList for *A_ARG_TYPE_BandwidthTestStatus* 40

Table 2-10: allowedValueList for *A_ARG_TYPE_Interfaces* 43

Table 2-11: allowedValueList for *A_ARG_TYPE_InterfaceResetStatus* 44

Table 2-12: allowedValueList for *A_ARG_TYPE_LogLevel*[1] 45

Table 2-13: Event Moderation 47

Table 2-14: Allowed Values for *Announcement.dm.upnp.org field-value* 48

Table 2-15: Actions 49

Table 2-16: Arguments for *Reboot()* 51

Table 2-17: Error Codes for *Reboot()* 52

Table 2-18: Error Codes for *BaselineReset()* 53

Table 2-19: Arguments for *GetDeviceStatus()* 53

Table 2-20: Error Codes for *GetDeviceStatus()* 54

Table 2-21: Arguments for *SetSequenceMode()* 54

Table 2-22: Error Codes for *SetSequenceMode()* 54

Table 2-23: Arguments for *GetSequenceMode()* 55

Table 2-24: Error Codes for *GetSequenceMode()* 55
Table 2-25: Arguments for *Ping()* 56
Table 2-26: Error Codes for *Ping()* 56
Table 2-27: Arguments for *GetPingResult()* 57
Table 2-28: Error Codes for *GetPingResult()* 57
Table 2-29: Arguments for *NSLookup()* 58
Table 2-30: Error Codes for *NSLookup()* 59
Table 2-31: Arguments for *GetNSLookupResult()* 59
Table 2-32: Error Codes for *GetNSLookupResult()* 60
Table 2-33: Arguments for *Traceroute()* 60
Table 2-34: Error Codes for *Traceroute()* 61
Table 2-35: Arguments for *GetTracerouteResult()* 61
Table 2-36: Error Codes for *GetTracerouteResult()* 62
Table 2-37: Arguments for *GetBandwidthTestInfo()* 62
Table 2-38: Error Codes for *GetBandwidthTestInfo()* 63
Table 2-39: Arguments for *BandwidthTest()* 63
Table 2-40: Error Codes for *BandwidthTest()* 64
Table 2-41: Arguments for *GetBandwidthTestResult()* 65
Table 2-42: Error Codes for *GetBandwidthTestResult()* 65
Table 2-43: Arguments for *InterfaceReset()* 66
Table 2-44: Error Codes for *InterfaceReset()* 66
Table 2-45: Arguments for *GetInterfaceResetResult()* 67
Table 2-46: Error Codes for *GetInterfaceResetResult()* 68
Table 2-47: Arguments for *SelfTest()* 68
Table 2-48: Error Codes for *SelfTest()* 68
Table 2-49: Arguments for *GetSelfTestResult()* 69
Table 2-50: Error Codes for *GetSelfTestResult()* 69
Table 2-51: Arguments for *GetTestIDs()* 70
Table 2-52: Error Codes for *GetTestIIDs()* 70
Table 2-53: Arguments for *GetActiveTestIDs()* 70
Table 2-54: Error Codes for *GetActiveTestIIDs()* 71
Table 2-55: Arguments for *GetTestInfo()* 71

Table 2-56: Error Codes for *GetTestInfo()* 71
Table 2-57: Arguments for *CancelTest()* 72
Table 2-58: Error Codes for *CancelTest()* 72
Table 2-59: Arguments for *GetLogURIs()* 73
Table 2-60: Error Codes for *GetLogURIs()* 73
Table 2-61: Arguments for *SetLogInfo()* 73
Table 2-62: Error Codes for *SetLogInfo()* 74
Table 2-63: Arguments for *GetLogInfo()* 74
Table 2-64: Error Codes for *GetLogInfo()* 75
Table 2-65: Arguments for *GetACLData()* 75
Table 2-66: Error Codes for *GetACLData()* 76
Table 2-67: Common Error Codes 76
Table 2-68: Bandwidth Test Protocols 81
Table 2-69: Bandwidth Test Capabilities 82
Table 2-70: Bandwidth Test Settings 83
Table 2-71: Bandwidth Test Results 85
Table 2-72: Bandwidth Test HTTP and FTP Baseline Profile 88
Table 2-73: Bandwidth Test HTTP and FTP BBF Profile 88
Table 2-74: Bandwidth Test Echo and EchoPlus Baseline Profile 90
Table 2-75: Bandwidth Test Echo and EchoPlus BBF Profile 90
Table 2-76: Bandwidth Test Iperf Baseline Profile 91
Table 2-77: Bandwidth Test Discovered Devices 104

L f F g e

Figure 2-1: Bandwidth Test Overview 19
Figure 2-2: Test State Transition Diagram 29
Figure 2-3: Example *Parent Devices* 93
Figure 2-4: *RebootNow* Example 95
Figure 2-5: *RebootLater* Example 96
Figure 2-6: *SequenceMode* Example 99

1. O e e a d Sc e

This service definition is compliant with the UPnP Device Architecture version 1.0 [UDA1.0]. It defines a service type referred to herein as *BasicManagement:2* service or, where the version number is not significant, *BasicManagement* service.

1.1. I d c

This service provides basic management operations. It enables the following functions:

- Indication of overall device status.
- Performing maintenance actions such as rebooting.
- Running diagnostic tests such as an IP ping test, bandwidth test or self test.
- Enabling / disabling logging and retrieving log files.

The diagnostic test actions are treated as an optional *Diagnostics Feature*. When this feature is supported, some of the service's optional diagnostic test actions are required, e.g. *GetActiveTestIDs()* and *GetTestInfo()*. For each diagnostic test, a corresponding test-specific feature is defined, which includes the requirements of the *Diagnostics Feature* and additionally requires the actions associated with that test to be supported, e.g. the *Ping Diagnostics Feature* requires *GetActiveTestIDs()*, *GetTestInfo()* etc, and also *Ping()* and *GetPingResult()*.

Management operations can be protected by an optional *Security Feature* based on the *DeviceProtection:1* service [DPS]. Actions that do not return sensitive information, change the device configuration, or affect normal device operation can always be invoked by all control points. If the *Security Feature* is supported, other actions can only be invoked if the control point is appropriately authorized.

This specification frequently uses the term *Parent Device*. This refers to UPnP device/service sub-tree whose root is the UPnP device that contains the *BasicManagement* service instance. UPnP actions or other operations on a *Parent Device* SHOULD apply to all levels of this sub-tree, but SHOULD NOT apply to an embedded device that itself contains a *BasicManagement* service instance.

There are references to *Parent Device* start, stop, restart and/or reboot in several places. These mean the following:

- "*Parent Device* start" refers to *Parent Device* startup, including the sending out of _____ messages.
- "*Parent Device* stop" refers to *Parent Device* shutdown, including (if possible) the sending out of _____ messages.
- "*Parent Device* restart" refers to "*Parent Device* stop" followed by "*Parent Device* start".
- "*Parent Device* reboot" refers to "*Parent Device* stop" followed by a reboot of the targeted Execution Environment and/or the Operating System, followed by "*Parent Device* start". For discussion of whether the targeted Execution Environment and/or the Operating System will be rebooted, see the description of the *Reboot()* action in section 2.5.1.

1.2. Refe e ce

This section lists the normative references used in the UPnP DM specifications and includes the tag inside square brackets that is used for each such reference:

[CMS] *UPnP ConfigurationManagement:2 Service Document*, UPnP Forum, February 16, 2012, http://www.upnp.org/specs/dm/UPnP-dm-ConfigurationManagement-v2-Service.pdf.

[CMS-XSD] *XML Schema for ConfigurationManagement:2*, UPnP Forum, February 16, 2012, http://www.upnp.org/schemas/dm/cms-v2.xsd.

[DEVICE] *UPnP ManageableDevice:2 Device Document*, UPnP Forum, February 16, 2012,http://www.upnp.org/specs/dm/UPnP-dm-ManageableDevice-v2-Device.pdf.

[DNS] *RFC 1035, Domain Names - Implementation and Specification*, IETF, November 1987, http://tools.ietf.org/html/rfc1035

[DPS] *UPnP DeviceProtection:1 Service Document*, UPnP Forum, February 24, 2011, http://upnp.org/specs/gw/UPnP-gw-DeviceProtection-v1-Service.pdf.

[DSCP] *RFC 2474, Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*, IETF, December 1988, http://tools.ietf.org/html/rfc2474

[EBNF] *W3C Extensible Markup Language (XML) 1.0 (Fifth Edition) -Notation section, http://www.w3.org/TR/REC-xml#sec-notation*

[ECHO] *RFC 862, Echo Protocol*, IETF, May 1983, http://tools.ietf.org/html/rfc862

[FTP] *RFC 959, File Transfer Protocol (FTP)*, IETF, October 1985, http://tools.ietf.org/html/rfc959

[HTTP] *RFC 2616, Hypertext Transfer Protocol – HTTP/1.1*, IETF, June 1999, http://tools.ietf.org/html/rfc2616

[ICMP] *RFC 792, Internet Control Message Protocol*, IETF, September 1981, http://tools.ietf.org/html/rfc792

[IPERF] *A tool to measure the bandwidth and quality of a network link*, http://sourceforge.net/projects/iperf (v2), http://code.google.com/p/iperf (v3), http://openmaniak.com/iperf.php (tutorial)

[REQLEV] *RFC 2119, Key words for use in RFCs to Indicate Requirement Levels*, IETF, March 1997, http://tools.ietf.org/html/rfc2119

[SMS] *UPnP SoftwareManagement:2 Service Document*, UPnP Forum, February 16, 2012, http://www.upnp.org/specs/dm/UPnP-dm-SoftwareManagement-v2-Service.pdf.

[SYSLOG] *RFC 3164, The BSD syslog Protocol*, IETF, August 2001, http://tools.ietf.org/html/rfc3164

[TERMS] *RFC 4689, Terminology for Benchmarking Network-layer Traffic Control Mechanisms*, IETF, October 2006, http://tools.ietf.org/html/rfc4689

[TR-143] *TR-143, Enabling Network Throughput Performance Tests and Statistical Monitoring,* Broadband Forum, 2008, http://www.broadband-forum.org/technical/download/TR-143_Corrigendum-1.pdf

[UDA1.0] UPnP Device Architecture version 1.0, UPnP Forum, October 2008, http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0.pdf

[URI] *RFC 3986, Uniform Resource Identifier (URI): Generic Syntax*, IETF, January 2005, http://tools.ietf.org/html/rfc3986

[XML]	*Extensible Markup Language (XML) 1.0 (Third Edition)*, W3C, February 2004, http://www.w3.org/TR/2004/REC-xml-20040204
[XML-NS]	*The "xml:" Namespace*, W3C, April 2006, http://www.w3.org/XML/1998/namespace
[XML-NMSP]	*Namespaces in XML*, W3C, August 2006, http://www.w3.org/TR/REC-xml-names
[XML-SCHEMA-1]	*XML Schema Part 1: Structures Second Edition*, W3C, October 2004, http://www.w3.org/TR/xmlschema-1
[XML-SCHEMA-2]	*XML Schema Part 2: Datatypes Second Edition*, W3C, October 2004, http://www.w3.org/TR/xmlschema-2
[XPATH-1.0]	*XML Path Language (XPath) Version 1.0*, W3C, November 1999, http://www.w3.org/TR/xpath

1.3. G a

ACL	Access Control List
BMS	*BasicManagement* Service
BNF	Backus Naur Form
CMS	*ConfigurationManagement* Service
SMS	*SoftwareManagement* Service
CSV	Comma Separated Value
DM	Device Management
XSD	XML Schema Definition

1.4. N a

- In this document, features are described as Required, Recommended, or Optional as follows:

  The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" in this specification are to be interpreted as described in [REQLEV].

  In addition, the following keywords are used in this specification:

  PROHIBITED – The definition or behavior is an absolute prohibition of this specification. Opposite of REQUIRED.

  CONDITIONALLY REQUIRED – The definition or behavior depends on a condition. If the specified condition is met, then the definition or behavior is REQUIRED, otherwise it is PROHIBITED.

  CONDITIONALLY OPTIONAL – The definition or behavior depends on a condition. If the specified condition is met, then the definition or behavior is OPTIONAL, otherwise it is PROHIBITED.

These keywords are thus capitalized when used to unambiguously specify requirements over protocol and application features and behavior that affect the interoperability and security of implementations. When these words are not capitalized, they are meant in their natural-language sense.

- Strings that are to be taken literally are enclosed in "double quotes."
- Words that are emphasized are printed in *italic*.
- Data model names and values, and literal XML, are printed using the character style.
- Keywords that are defined by the UPnP DM Working Committee are printed using the *forum* character style.
- Keywords that are defined by the UPnP Device Architecture are printed using the ____ character style.
- A double colon delimiter, "::", signifies a hierarchical parent-child (parent::child) relationship between the two objects separated by the double colon. This delimiter is used in multiple contexts, for example: Service::Action(), Action()::Argument.

1.4.1. Da a T e

This specification uses data type definitions from two different sources. The UPnP Device Architecture defined data types are used to define state variable and action argument data types [UDA1.0]. The XML Schema namespace is used to define XML-valued action arguments [XML-SCHEMA-2] (including [CMS] data model parameter values).

For UPnP Device Architecture defined Boolean data types, it is strongly RECOMMENDED to use the value "_" for false, and the value "_" for true. However, when used as input arguments, the values "____", "__", "____", "___" may also be encountered and MUST be accepted. Nevertheless, it is strongly RECOMMENDED that all state variables and output arguments be represented as "_" and "_".

For XML Schema defined Boolean data types, it is strongly RECOMMENDED to use the value "*0*" for false, and the value "*1*" for true. However, when used within input arguments, the values "*false*", "*true*" may also be encountered and MUST be accepted. Nevertheless, it is strongly RECOMMENDED that all XML Boolean values be represented as "*0*" and "*1*".

XML elements that are of type (for example [CMS] data model parameter values) MUST include an attribute that indicates the actual data type of the element value. This is a SOAP requirement.

1.4.2. S g E bedded O he S g

Some string variables, arguments and other XML elements and attributes (including [CMS] data model parameter values) described in this document contain substrings that MUST be independently identifiable and extractable for other processing. This requires the definition of appropriate substring delimiters and an escaping mechanism so that these delimiters can also appear as ordinary characters in the string and/or its independent substrings.

This document uses such embedded strings in Comma Separated Value (CSV) lists (see section 1.5.1). Escaping conventions use the backslash character, "\" (character code U+005C), as follows:

a) Backslash ("\") is represented as "\\".

b) Comma (",") is represented as "\," in individual substring entries.

c) Double quote (""") is not escaped.

This document also uses such embedded strings to represent XML documents (see section 1.5.2). Escaping conventions use XML entity references as specified in [XML] Section 2.4. For example:

a) Ampersand ("&") is represented as " " or via a numeric character reference.

b) Left angle bracket ("<") is represented as " " or via a numeric character reference.

c) Right angle bracket (">") usually doesn't have to be escaped, but often is, in which case it is represented as " " or via a numeric character reference.

1.5. De ed Da a T e

This section defines a derived data type that is represented as a string data type with special syntax. This specification uses string data type definitions that originate from two different sources. The UPnP Device Architecture defined _____ data type is used to define state variable and action argument string data types. The XML Schema namespace is used to define data types. The following definition applies to both string data types.

1.5.1. C a Se a a ed Va e (CSV) L

The UPnP DM services use state variables, action arguments and other XML elements and attributes that represent lists – or one-dimensional arrays – of values. [UDA1.0] does not provide for either an array type or a list type, so a list type is defined here. Lists MAY either be homogeneous (all values are the same type) or heterogeneous (values of different types are allowed). Lists MAY also consist of repeated occurrences of homogeneous or heterogeneous subsequences, all of which have the same syntax and semantics (same number of values, same value types and in the same order).

- The data type of a homogeneous list is _____ or and denoted by CSV (x), where x is the type of the individual values.
- The data type of a heterogeneous list is also _____ or and denoted by CSV (w, x [, y, z]), where w, x, y and z are the types of the individual values, and the square brackets indicate that y and z (and the preceding comma) are optional. If the number of values in the heterogeneous list is too large to show each type individually, that variable type is represented as CSV (*heterogeneous*), and the variable description includes additional information as to the expected sequence of values appearing in the list and their corresponding types. The data type of a repeated subsequence list is _____ or and denoted by CSV ({w, x, y, z}), where w, x, y and z are the types of the individual values in the subsequence and the subsequence MAY be repeated zero or more times (in this case none of the values are optional).

The individual value types are specified as [UDA1.0] data types or _____ data types for _____ lists, and as [XML-SCHEMA-2] data types for lists.

- A list is represented as a _____ type (for state variables and action arguments) or type (within other XML elements and attributes).
- Commas separate values within a list.
- Integer values are represented in CSVs with the same syntax as the integer data type specified in [UDA1.0] (that is: optional leading sign, optional leading zeroes, numeric ASCII).
- Boolean values are represented in state variable and action argument CSVs as either "_" for false or "_" for true. These values are a subset of the defined Boolean data type values specified in [UDA1.0]: _, ____, __, _, ____, ___.

- Boolean values are represented in other XML element CSVs as either “ ” for false or “ ” for true. These values are a subset of the defined Boolean data type values specified in [XML-SCHEMA-2]: , , , .
- Escaping conventions for the comma and backslash characters are defined in section 1.4.2.
- The number of values in a list is the number of unescaped commas, plus one. The one exception to this rule is that an empty string represents an empty list. This means that there is no way to represent a list consisting of a single empty string value.
- White space before, after, or interior to any numeric data type is not allowed.
- White space before, after, or interior to any other data type is part of the value.

Tab e 1-1: CSV E a e

T e ef e e f Va e g		C e
CSV (_____)	“first,second”	List of 2 strings used as state variable or action argument value.
CSV ( )	“first,second”	List of 2 strings used within an XML element
CSV ( )	“first, second ”	List of 2 strings used within an XML element. Each element is of type so, even though the second value is “ second ” and has leading and trailing spaces, the value seen by the application will be “second” because collapses whitespace.
CSV (_____, _____ _____ , _____])	“Warning,2009-07-07T13:22:41, third\,value”	List of _____, _____ and (optional) _____ used as state variable or action argument value. Note the leading space and escaped comma in the third value, which is “ third,value”.
CSV (_____, _____ _____ , _____])	“Warning,2009-07-07T13:22:41,”	As above but third value is empty.
CSV (_____, _____ _____ , _____])	“Warning,2009-07-07T13:22:41”	As above but third value is omitted.
CSV (*A_ARG_TYPE_-Host*)	“grumpy,sleepy”	List of data items used as action argument value, each of which obeys the rules governing *A_ARG_TYPE_Host*. Any comma or backslash characters within a data item would have been escaped.
CSV (__)	“1, 2”	Illegal CSV. White space is not allowed as part of an integer value.
CSV (_____)	“a,,c,”	List of 4 strings “a”, “”, “c” and “”.

T e ef e e f Va e g		C e
CSV (_____)	“”	Empty list. It is not possible to create a list containing a single empty string.

1.5.2. E bedded XML D c e

An XML document is a string that represents a valid XML 1.0 document according to a specific schema. Every occurrence of the phrase “*XML Document*” is italicized and preceded by the document’s root element name (also italicized), as listed in column 3, “Valid Root Element(s)” of Table 1-3, “Schema-related Information”. For example, the phrase *SupportedDataModels XML Document* refers to a valid XML 1.0 document according to the CMS schema [CMS-XSD]. Such a document comprises a single root element, optionally preceded by the XML declaration

.

This string will therefore be of one of the following two forms:

“ ”

or

“ ”

Escaping conventions for the ampersand, left angle bracket and right angle bracket characters are defined in section 1.4.2.

For consistency with [UDA1.0] and for future extensibility, devices and control points MUST ignore the following in embedded XML documents:

- Any unknown XML elements and their sub elements or content,
- Any unknown attributes and their values,
- Any XML comments that they do not understand, and
- Any XML processing instructions that they do not understand.

1.6. Ma age e f XML Na e ace S a da d ed DCP

UPnP specifications make extensive use of XML namespaces. This allows separate DCPs, and even separate components of an individual DCP, to be designed independently and still avoid name collisions when they share XML documents. Every name in an XML document belongs to exactly one namespace. In documents, XML names appear in one of two forms: qualified or unqualified. An unqualified name (or no-colon-name) contains no colon (“ ”) characters. An unqualified name belongs to the document’s default namespace. A qualified name is two no-colon-names separated by one colon character. The no-colon-name before the colon is the qualified name’s namespace prefix, the no-colon-name after the colon is the qualified name’s “local” name (meaning local to the namespace identified by the namespace prefix). Similarly, the unqualified name is a local name in the default namespace.

The formal name of a namespace is a URI. The namespace prefix used in an XML document is not the name of the namespace. The namespace name is globally unique. It has a single definition that is accessible to anyone who uses the namespace. It has the same meaning anywhere that it is used, both inside and outside XML documents. The namespace prefix, however, in formal XML usage, is defined only in an XML document. It must be locally unique to the document. Any valid XML no-colon-name may be used. And, in formal XML usage, no two XML documents are ever required to use the same namespace prefix to

refer to the same namespace. The creation and use of the namespace prefix was standardized by the W3C XML Committee in [XML-NMSP] strictly as a convenient local shorthand replacement for the full URI name of a namespace in individual documents.

All of the namespaces used in this specification are listed in the Tables "Namespace Definitions" and "Schema-related Information". For each such namespace, Table 1-2, "Namespace Definitions" gives a brief description of it, its name (a URI) and its defined "standard" prefix name. Some namespaces included in these tables are not directly used or referenced in this document. They are included for completeness to accommodate those situations where this specification is used in conjunction with other UPnP specifications to construct a complete system of devices and services. The individual specifications in such collections all use the same standard prefix. The standard prefixes are also used in Table 1-3, "Schema-related Information", to cross-reference additional namespace information. This second table includes each namespace's valid XML document root element(s) (if any), its schema file name, versioning information (to be discussed in more detail below), and a link to the entry in Section 1.2, "References" for its associated schema.

The normative definitions for these namespaces are the documents referenced in Table 1-3. The schemas are designed to support these definitions for both human understanding and as test tools. However, limitations of the XML Schema language itself make it difficult for the UPnP-defined schemas to accurately represent all details of the namespace definitions. As a result, the schemas will validate many XML documents that are not valid according to the specifications.

Table 1-2: Namespace Definitions

Standard Namespace Prefix	Namespace Name	Namespace Description	Name Definition Document Reference
DM Working Committee defined namespaces			
	urn:schemas-upnp-org:dm:bms	BMS data structures	[BMS]
	urn:schemas-upnp-org:dm:cms	CMS data structures	[CMS]
	urn:schemas-upnp-org:dm:sms	SMS data structures	[SMS]
[1]	urn:schemas-upnp-org:dm:bms:nsl	BMS NSLookupResult	[BMS]
Externally defined namespaces			
	http://www.w3.org/2001/XMLSchema	XML Schema Language 1.0	[XML-SCHEMA-1] [XML-SCHEMA-2]
	http://www.w3.org/2001/XMLSchema-instance	XML Schema Instance Document schema	Sections 2.6 & 3.2.7 of [XML-SCHEMA-1]
	http://www.w3.org/XML/1998/namespace	The "xml:" Namespace	[XML-NS]

[1] was defined in *BasicManagement:1* and remains valid. Its definitions are also present in the namespace, which is where any future enhancements will be made.

Table 1-3: Schema-related Information

Standard Namespace Prefix	Relative URI and File Name[1] • File 1, 2, 3	Valid Reference(s)	Schema Reference
DM Working Committee defined namespaces			
	• bms-v*n*-*yyyymmdd*.xsd • bms-v*n*.xsd • bms.xsd		[BMS]
	• cms-v*n*-*yyyymmdd*.xsd • cms-v*n*.xsd • cms.xsd		[CMS]
	• sms-v*n*-*yyyymmdd*.xsd • sms-v*n*.xsd • sms.xsd		[SMS]
[2]	• bmsnsl-v*n*-*yyyymmdd*.xsd • bmsnsl-v*n*.xsd • bmsnsl.xsd		[BMS]

[1] Absolute URIs are generated by prefixing the relative URIs with "http://www.upnp.org/schemas/dm/".

[2] was defined in *BasicManagement:1* and remains valid. Its definitions are also present in the namespace, which is where any future enhancements will be made.

1.6.1. Na e ace Na e , Na e ace Ve g a d Sche a Ve g

The UPnP DM service specifications define several data structures (such as state variables and action arguments) whose format is an XML instance document that must comply with one or more specific XML namespaces. Each namespace is uniquely identified by an assigned namespace name. The namespaces that are defined by the DM Working Committee MUST be named by a URN. See Table 1-2 "Namespace Definitions" for a current list of namespace names. Additionally, each namespace corresponds to an XML schema document that provides a machine-readable representation of the associated namespace to enable automated validation of the XML (state variable or action parameter) instance documents.

Within an XML schema and XML instance document, the name of each corresponding namespace appears as the value of an attribute within the root element. Each attribute also includes a namespace prefix that is associated with that namespace in order to disambiguate (a.k.a. qualify) element and attribute names that are defined within different namespaces. The schemas that correspond to the listed namespaces are identified by URI values that are listed in the attribute also within the root element. (See Section 1.6.2)

In order to enable both forward and backward compatibility, namespace names are permanently assigned and MUST NOT change even when a new version of a specification changes the definition of a namespace. However, all changes to a namespace definition MUST be backward-compatible. In other words, the updated definition of a namespace MUST NOT invalidate any XML documents that comply with an earlier definition of that same namespace. This means, for example, that a namespace MUST NOT be changed so that a new element or attribute is required. Although namespace names MUST NOT change, namespaces still have version numbers that reflect a specific set of definitional changes. Each time the definition of a namespace is changed, the namespace's version number is incremented by one.

Each time a new namespace version is created, a new XML schema document (.xsd) is created and published so that the new namespace definition is represented in a machine-readable form. Since an XML schema document is just a representation of a namespace definition, translation errors can occur. Therefore, it is sometime necessary to re-release a published schema in order to correct typos or other namespace representation errors. In order to easily identify the potential multiplicity of schema releases for the same namespace, the URI of each released schema MUST conform to the following format (called Form 1):

Form 1: "http://www.upnp.org/schemas/dm/" ***schema-root-name*** "-v" ***ver*** "-" ***yyyymmdd*** ".xsd"

where

- ***schema-root-name*** is the name of the root element of the namespace that this schema represents.
- ***ver*** corresponds to the version number of the namespace that is represented by the schema.
- ***yyyymmdd*** is the year, month and day (in the Gregorian calendar) that this schema was released.

Table 1-3 "Schema-related Information" identifies the URI formats for each of the namespaces that are currently defined by the UPnP DM Working Committee.

As an example, the original schema URI for the " " namespace might be "http://www.upnp.org/schemas/dm/cms-v1-20091231.xsd". If the UPnP DM service specifications were subsequently updated in the year 2010, the URI for the updated version of the " " namespace might be "http://www.upnp.org/schemas/dm/cms-v2-20100906.xsd".

In addition to the dated schema URIs that are associated with each namespace, each namespace also has a set of undated schema URIs. These undated schema URIs have two distinct formats with slightly different meanings:

Form 2: "http://www.upnp.org/schemas/dm/" ***schema-root-name*** "-v" ***ver*** ".xsd"

Form 3: "http://www.upnp.org/schemas/dm/" ***schema-root-name*** ".xsd"

Form 2 of the undated schema URI is always linked to the most recent release of the schema that represents the version of the namespace indicated by ***ver***. For example, the undated URI "…/dm/cms-v2.xsd" is linked to the most recent schema release of version 2 of the " " namespace. Therefore, on September 06, 2010 (20100906), the undated schema URI might be linked to the schema that is otherwise known as "…/dm/cms-v2-20100906.xsd". Furthermore, if the schema for version 2 of the " " namespace was ever re-released, for example to fix a typo in the 20100906 schema, then the same undated schema URI ("…/dm/cms-v2.xsd") would automatically be updated to link to the updated version 2 schema for the " " namespace.

Form 3 of the undated schema URI is always linked to the most recent release of the schema that represents the highest version of the namespace that has been published. For example, on December 31, 2009 (20091231), the undated schema URI "…/dm/cms.xsd" might be linked to the schema that is otherwise known as "…/dm/cms-v1-20091231.xsd". However, on September 06, 2010 (20100906), that same undated schema URI might be linked to the schema that is otherwise known as "…/dm/cms-v2-

20100906.xsd". When referencing a schema URI within an XML instance document or a referencing XML schema document, the following usage rules apply:

- All instance documents, whether generated by a service or a control point, MUST use Form 3.
- All UPnP DM published schemas that reference other UPnP DM schemas MUST also use Form 3.

Within an XML instance document, the definition for the attribute comes from the XML Schema namespace "http://www.w3.org/2002/XMLSchema-instance". A single occurrence of the attribute can declare the location of one or more schemas. The attribute value consists of a whitespace separated list of values that is interpreted as a namespace name followed by its schema location URL. This pair-sequence is repeated as necessary for the schemas that need to be located for this instance document.

In addition to the schema URI naming and usage rules described above, each released schema MUST contain a attribute in the root element. Its value MUST correspond to the format:

ver "-" ***yyyymmdd*** where ***ver*** and ***yyyymmdd*** are described above.

The attribute provides self-identification of the namespace version and release date of the schema itself. For example, within the original schema released for the " " namespace (…/cms-v1-20091231.xsd), the root element might contain the following attribute:

.

1.6.2. Na e ace U age E a e

The attribute for XML instance documents comes from the XML Schema instance namespace "http:://www.w3.org/2001/XMLSchema-instance". A single occurrence of the attribute can declare the location of one or more schemas. The attribute value consists of a whitespace separated list of values: namespace name followed by its schema location URL. This pair-sequence is repeated as necessary for the schemas that need to be located for this instance document.

Example 1:

Sample CMS XML Instance Document. Note that the references to the UPnP DM schemas do not contain any version or release date information. In other words, the references follow Form 3 from above. Consequently, this example is valid for all releases of the UPnP DM service specifications.

1.7. Ve d -def ed E e

Whenever vendors create additional vendor-defined state variables, actions or other XML elements and attributes, their assigned names and XML representation MUST follow the naming conventions and XML rules as specified in [UDA1.0], Section 2.5, "Description: Non-standard vendor extensions".

2. Se ce M de g Def

2.1. Se ceT e

The following service type identifies a service that is compliant with this template:

BasicManagement:2

2.2. Ke C ce

Basic management operations fall into three categories: maintenance, diagnostic tests, and logging. Security settings affect which control points are permitted to perform which basic management operations.

2.2.1. Ma e a ce

Two actions allow reboot and baseline reset of a *Parent Device* (see section 1.1) and possibly of the targeted Execution Environment and/or Operating System.

A third action can be used to indicate that a control point is planning to perform a sequence of actions, and requests a *Parent Device* to perform them as efficiently as possible.

2.2.2. D ag c e

The diagnostic test actions are treated as an optional *Diagnostics Feature*. When this feature is supported, some of the service's optional diagnostic test actions are required, e.g. *GetActiveTestIDs()* and *GetTestInfo()*. For each diagnostic test, a corresponding test-specific feature is defined, which includes the requirements of the *Diagnostics Feature* and additionally requires the actions associated with that test to be supported, e.g. the *Ping Diagnostics Feature* requires *GetActiveTestIDs()*, *GetTestInfo()* etc, and also *Ping()* and *GetPingResult()*.

Diagnostic tests are expected to take a significant length of time to execute. In particular, it cannot be assumed that a given test will complete within the 30 second response time allowed by [UDA1.0]. Therefore each diagnostic test has an action to request the test and another action to return the test result. Test request actions all return a unique test ID that can be used to determine the state of, or to cancel, the corresponding test.

Tests SHOULD be performed as soon as possible after they are successfully requested. However, it is up to the service implementation to decide when a given test can be performed. If a test has been successfully requested but cannot currently be performed, e.g. because it requires a resource that is currently in use, the test will remain in the *Requested* state until it can be performed or is canceled.

Once a given test has been performed, the test ID MUST remain valid, and the test results MUST remain available, at least until another test of the same type is successfully requested, or until a *Parent Device* restart, e.g. as a result of a power cycle or use of the *Reboot()* action. An implementation MAY retain test IDs and test results across a *Parent Device* restart[2], or retain more than one set of results per test type. Test IDs MUST become invalid when the corresponding test results are discarded. Individual test definitions MAY state more stringent test ID retention requirements.

Each invocation of a diagnostic test action requests a new test, regardless of whether the test arguments are the same as those for an earlier successfully requested test.

2.2.2.1. Basic tests

The following basic diagnostic tests are provided:

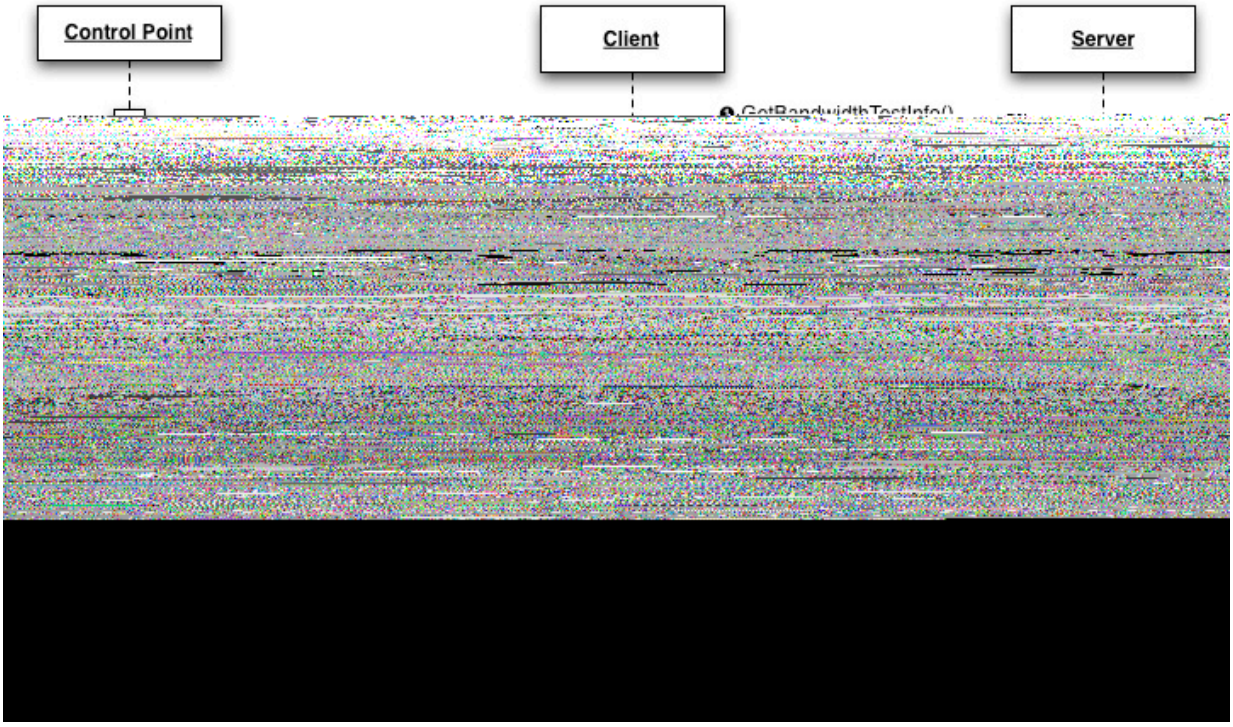
[2] An implementation that retains test IDs and test results across a *Parent Device* restart SHOULD provide a way of deleting this stored information, e.g. via a local GUI.

- *Ping()*, an IP-layer ping test.
- *NSLookup()*, an IP-layer DNS lookup test.
- *Traceroute()*, an IP-layer trace-route test.
- *InterfaceReset()*, an IP interface reset test.
- *SelfTest()*, an implementation-specific self-test.

2.2.2.2. Bandwidth tests

An extensible set of bandwidth tests can be invoked using the *GetBandwidthTestInfo()*, *BandwidthTest()* and *GetBandwidthTestResult()* actions.

When the client and server both support the *BandwidthTest()* action, it is usual to invoke it first on the server (to allow the server to prepare for the test) and then on the client (to initiate the test). This is illustrated (for an HTTP download test) in Figure 2-1. Refer to the numbered notes below the Figure for explanation.



The numbers below correspond to the white numbers in black circles in the Figure.

1. Control point calls *GetBandwidthTestInfo()* on the server (if supported) to discover server capabilities.
2. Control point calls *GetBandwidthTestInfo()* on the client (if supported) to discover client capabilities.
3. Control point decides which file to request from the server; control point allocates an opaque test session ID.
4. Control point calls *BandwidthTest()* on the server to warn it that the download will soon be happening, and to give it the test session ID. Server returns a test ID.

5. Control point calls *BandwidthTest()* on the client to initiate the download on the client. Client returns a test ID.
6. Client and server perform the download asynchronously.
7. Control point monitors or polls the test IDs. On completion:
8. Control point calls *GetBandwidthTestResult()* on the server to get server-side test result.
9. Control point calls *GetBandwidthTestResult()* on the client to get client-side test result.

If the server doesn't support *BasicManagement* (either because it is within the home network but doesn't support the service, or because it is outside the home network) the procedure is the same but the server interactions are omitted or use another protocol. From the *BasicManagement* point of view one would see the following.

1. Control point calls *GetBandwidthTestInfo()* on the client (if supported) to discover client capabilities.
2. Control point uses out of band knowledge of valid files to decide which file to request.
3. Control point calls *BandwidthTest()* on the client to initiate the test. Client returns a test ID.
4. Client and server perform the download test asynchronously.
5. Control point monitors or polls the test IDs. On completion:
6. Control point calls *GetBandwidthTestResult()* on the client to get client-side test result.

2.2.3. L gg g

A *Parent Device* can support multiple logs, and the list of logs can change at run-time. Each log is identified by a URI. Each log can be independently enabled / disabled, has an associated log level, and has a URL via which it can be retrieved.

The list of logs, log enable/disable status and log level MUST persist across a *Parent Device* restart. Log contents and URL MAY persist across a *Parent Device* restart.

Logs are of limited size and SHOULD behave as FIFOs. The maximum size of a given log is determined by the implementation and/or the UPnP working committee (or other organization) that specified it. As a guideline, it SHOULD be possible to log several hundreds of UPnP actions.

The UPnP Device Management working committee defines only a generic log configuration and retrieval mechanism. It does not define log formats. Other working committees MAY define DCP-specific logs, which MAY include format requirements. Each such DCP-specific log MUST be identified by a DCP-specified URN that is used as the URI in the log-related actions.

The first entry in the list of supported logs SHOULD be a *Parent Device*'s default (or primary) log.

Unless otherwise specified by a UPnP working committee, UPnP actions invoked on a *Parent Device* SHOULD be logged to the primary (or default) log. UPnP action-related log entries SHOULD identify the control point, the action and the action's outcome (success / failure).

The list of log URIs, and related information such as enable/disable status, log level and URL, is not regarded as sensitive information. Any security requirements pertain to the out-of-band protocol via which the logs are retrieved. This out-of-band protocol is implied by the log's URL.

2.2.4. Sec

Actions that do not return sensitive information, change the device configuration, or affect normal device operation are referred to as *Non-Restrictable* actions and can always be invoked by all control points.

All other actions are referred to as *Restrictable* actions. If the optional *Security Feature* (based on the *DeviceProtection:1* service [DPS]) is not supported, all actions can be invoked by all control points. If the *Security Feature* is supported, *Restrictable* actions can only be invoked if the control point is appropriately authorized. Table 2-15 specifies which actions are *Non-Restrictable* and *Restrictable*.

The terms *Role List* and *Restricted Role List* are defined by *DeviceProtection:1*. Each action has an associated *Role List*; a control point that possesses a *Role* in the *Role List* can unconditionally invoke the action. Some actions also have a *Restricted Role List*; a control point that does not possess a *Role* in the *Role List* but does possess a *Role* in the *Restricted Role List* might be able to invoke the action (it's up to the action definition to specify this).

The *Public Role* is defined by *DeviceProtection:1*. All control points automatically possess the *Public Role*, and all control points can unconditionally invoke all actions that have a *Role List* of "*Public*". Therefore:

- If the *Security Feature* is not supported, behavior is the same as if the feature was supported and all actions had a *Role List* of "*Public*" and an empty *Restricted Role List*.
- Regardless of whether or not the *Security Feature* is supported, all *Non-Restrictable* actions have a *Role List* of "*Public*" and an empty *Restricted Role List*.

For *Restrictable* actions, this specification defines RECOMMENDED values for the *Role Lists* and *Restricted Role Lists*. Device manufacturers are permitted to choose different values.

2.2.4.1. _____ access control list

The previous section explained that, when the *Security Feature* is supported, *Restrictable* actions can have *Restricted Role Lists* and that, for such actions, a control point that does not possess a *Role* in the *Role List* but does possess a *Role* in the *Restricted Role List* might be able to invoke the action. This decision is made by consulting the *BasicManagement:2* access control list.

Section 2.3.37 specifies *BasicManagement:2* access control list syntax. To re-iterate, this access control list is relevant only for *Restrictable* actions, and only for control points that do not possess *Roles* in the *Role List* but do possess *Roles* in the *Restricted Role List*.

- If there is no access control entry for a given action, the access control decision is made as described in the description of the action in question. For example, the description of the *CancelTest()* action states that the control point that initiated a test can always cancel it, and this rule applies regardless of whether the access control list contains an entry for *CancelTest()*.
- If there are one or more access control entries for a given action, access is permitted if the conditions stated in the action description are met, and if any of the action's filters match (and the control point possesses a *Role* in the corresponding *Role List*).

For example, given the example access control list from section 2.3.37:

…and assuming that the action *Role Lists* and *Restricted Role Lists* have the values recommended in Table 2-15, the following access control decisions would be made:

- A control point possessing the *Admin* *Role* will be unconditionally permitted to invoke all actions (because the *Role List* permits this).
- A control point possessing the *Basic* *Role* (but not the *Admin* *Role*) will be unconditionally permitted to invoke the *Ping()* action (because the *Role List* permits this) but will be permitted to invoke the *InterfaceReset()* action only if the *Interfaces* argument has the value "*RequestInterface*" (because the access control list specifies this).
- A control point possessing the *Basic* *Role* (but not the *Admin* *Role*) will be unconditionally permitted to invoke the *GetLogInfo()* action (because the *Role List* permits this) but will be permitted to invoke the *SetLogInfo()* action only if the *LogURI* parameter contains the string ":restricted:" (because the access control list specifies this).
- A control point possessing the *Public* *Role* (but not the *Basic* or *Admin* *Role*) will be permitted to invoke the *Ping()* action if the action is invoked over a TLS session and the control point *Identity* is present in the *DeviceProtection:1* ACL (because the *Ping()* action description states this). The same control point will be permitted to cancel any tests that it initiated (because the *CancelTest()* description states this).

A control point possessing the *Public* *Role* (but not the *Basic* or *Admin* *Role*) will never be permitted to invoke the *Reboot()* or *BaselineReset()* actions (because the *Role List* does not permit this and *Public* is not in the *Restricted Role List*).

Note that it is never necessary for an access control list entry to include the *Admin* *Role* in its *Role List*, because, as stated in [DEVICE] section 2.4.4, the *Admin* *Role* always grants full access to all actions.

2.3. S a e Va ab e

Reader Note: For a first-time reader, it may be more helpful to read the action definitions before reading the state variable definitions.

Tab e 2-1: S a e Va ab e

Va ab e Na e	Re . O .[1]	Da a T e	A ed Va e	Defa Va e	E g. U
DeviceStatus	*R*	_____	CSV[2] (_____, _____ [, _____]), where first string is: *OK, Warning, Error, Fatal* (section 2.3.1)		
SequenceMode	*O*	_____	(section 2.3.2)	"_"	
TestIDs	*CR*[3]	_____	CSV (*A_ARG-_TYPE_TestID*) (section 2.3.3)	""	

Variable Name	Req. or Opt.[1]	Data Type	Allowed Value	Default Value	Eg. Unit
ActiveTestIDs	CR[3]	string	CSV (A_ARG_TYPE_TestID) (section 2.3.4)	""	
LogURIs	O	string	CSV (A_ARG_TYPE_Log-URI) (section 2.3.5)		
A_ARG_TYPE_Boolean	O	boolean	(section 2.3.6)		
A_ARG_TYPE_String	O	string	(section 2.3.7)		
A_ARG_TYPE_UShort	O	ui2	(section 2.3.8)		
A_ARG_TYPE_UInt	O	ui4	(section 2.3.9)		
A_ARG_TYPE_DateTime	O	dateTime	(section 2.3.10)		
A_ARG_TYPE_MSecs	O	ui4	(section 2.3.11)		Milliseconds
A_ARG_TYPE_RebootStatus	O	string	RebootNow, RebootLater (section 2.3.12)		
A_ARG_TYPE_TestID	O	ui4	(section 2.3.13)		
A_ARG_TYPE_TestType	O	string	BandwidthTest, InterfaceReset, NSLookup, Ping, SelfTest, Traceroute (section 2.3.14)		
A_ARG_TYPE_TestState	O	string	Requested, InProgress, Canceled, Completed (section 2.3.15)		
A_ARG_TYPE_TestEndpoint	O	string	Server, Client (section 2.3.16)		
A_ARG_TYPE_TestSchedule	O	string	(section 2.3.17)		
A_ARG_TYPE_TestSessID	O	string	(section 2.3.18)		
A_ARG_TYPE_DSCP	O	ui4	Between 0 and 63 inclusive (section 2.3.19)		
A_ARG_TYPE_Host	O	string	(section 2.3.20)		

Variable Name	Req. or Opt.[1]	Data Type	Allowed Value	Default Value	Eg. U
A_ARG_TYPE_Hosts	O		CSV (A_ARG_TYPE_Host) (section 2.3.21)		
A_ARG_TYPE_HostName	O		(section 2.3.22)		
A_ARG_TYPE_PingStatus	O		Success, Error_CannotResolveHostName (section 2.3.23)		
A_ARG_TYPE_NSLookupStatus	O		Success, Error_DNSServerNotResolved (section 2.3.24)		
A_ARG_TYPE_NSLookupResult	O		(section 2.3.25)		
A_ARG_TYPE_TracerouteStatus	O		Success, Error_CannotResolveHostName, Error_MaxHopCountExceeded (section 2.3.26)		
A_ARG_TYPE_BandwidthTestInfo	O		(section 2.3.27)		
A_ARG_TYPE_BandwidthTestSpec	O		(section 2.3.28)		
A_ARG_TYPE_BandwidthTestStatus	O		(section 2.3.29)		
A_ARG_TYPE_BandwidthTestResult	O		(section 2.3.30)		
A_ARG_TYPE_Interfaces	O		AllInterfaces, RequestInterface (section 2.3.31)		
A_ARG_TYPE_InterfaceResetStatus	O		Success, Error (section 2.3.32)		
A_ARG_TYPE_LogURI	O		(section 2.3.33)		
A_ARG_TYPE_LogURL	O		(section 2.3.34)		
A_ARG_TYPE_LogLevel	O		Emergency, Alert, Critical, Error, Warning, Notice, Informational, Debug (section 2.3.35)		

Va ab e Na e	Re . O .[1]	Da a T e	A ed Va e	Defa Va e	E g. U
A_ARG_TYPE_LogMaxSize	*O*	___	(section 2.3.36)		Bytes
A_ARG_TYPE_ACL	*CR*[4]	_____	(section 2.3.37)		
Non-standard state variables implemented by an UPnP vendor go here.	*X*	*TBD*	*TBD*	*TBD*	*TBD*

[1] *R* = REQUIRED, *O* = OPTIONAL, *CR* = CONDITIONALLY REQUIRED, *X* = Non-standard.

[2] CSV stands for Comma-Separated Value list. The type between brackets denotes the UPnP data type used for the elements inside the list (section 1.5.1).

[3] REQUIRED if the *Diagnostics Feature* is supported.

[4] REQUIRED if the *Security Feature* is supported.

2.3.1. *DeviceStatus*

Indicates the *Parent Device* status, the date/time at which it last changed, and additional optional information.

This is a CSV (_____, _____ [, _____]) list (section 1.5.1):

- The first value is the *Parent Device* status and MUST be one of *OK*, *Warning*, *Error* or *Fatal*.
- The second value is the date/time at which the *Parent Device* started up or its status last changed (whichever occurred most recently). This value MUST obey the rules specified for *A_ARG_TYPE_DateTime* (section 2.3.10).
- The optional third value is a vendor-specific string that can give additional information about the *Parent Device* status.

For example:

- OK,2009-06-15T12:00:00
- Warning,2009-06-15T13:00:00,More hints\, info etc about this warning

In the second example, if the final comma had not been escaped the CSV list would have been illegal because it would have had four (rather than three) values, and the third value would have been "More hints" rather than "More hints, info etc about this warning".

2.3.2. *SequenceMode*

Indicates whether a control point is currently executing a sequence of actions. The value of *SequenceMode* MAY persist across *Parent Device* restarts.

A *SequenceMode* "_" → "_" transition (via *SetSequenceMode("_")*):

- Indicates that a control point is planning to execute a sequence of actions, and requests the *Parent Device* to perform them as efficiently as possible.
- Initializes a conceptual countdown timer to 60 seconds. This timer can be re-initialized to 60 seconds at any time by again invoking *SetSequenceMode("_")* or by invoking any action whose

behavior can be affected by *SequenceMode*. When the timer expires, *SequenceMode* is automatically set to "_".

- If an action's behavior can be affected by *SequenceMode*, this MUST be specified in the action description. Therefore, if an action's description makes no mention of *SequenceMode*, then that action's behavior is not affected by *SequenceMode*.

- If an action's behavior can be affected by *SequenceMode*, this special behavior occurs only if *SequenceMode* is "_" when the action is invoked.

A *SequenceMode* "_" → "_" transition (either via *SetSequenceMode("_")* or via timer expiry as described above):

- Indicates that all the actions of a sequence executed while *SequenceMode* was "_" have now been invoked.

- Requests that the *Parent Device* MUST as soon as possible apply any changes that were not applied while *SequenceMode* was "_", and complete any operations that were not performed while *SequenceMode* was "_".

A *SequenceMode* value of "_" also serves to inform control points whether another control point is currently executing a sequence of actions.

For example, a given platform might behave as follows:

- : commits changes to a "pending" configuration, which has to be copied to the "running" configuration in order to be applied. The copy to the "running" configuration could require a reboot.

  SequenceMode = "_" would allow a set of changes spanning multiple actions to be applied all at once. In other words, the device would wait until all the requested changes had been received before attempting to copy them from "pending" to "running" and performing the reboot.

- : disables UPnP control interface before downloading file, then reboots.

  SequenceMode = "_" would allow multiple deployment units to be installed before rebooting.

The *Parent Device* SHOULD if at all possible honor the above *SequenceMode* behavior, but it is understood that there might be exceptional circumstances under which it is unable to do so.

2.3.3. *TestIDs*

Comma-separated list of all known test IDs. This is a CSV (*A_ARG_TYPE_TestID*) list.

A test is known if it has been successfully requested and has not yet been deleted. Therefore, a test ID MUST be added to *TestIDs* when its test is successfully requested, and it MUST be removed from *TestIDs* when the implementation decides to delete all information associated with the test.

See section 2.3.13 for a general description of test IDs. See Figure 2-2 for the test state transition diagram.

2.3.4. *ActiveTestIDs*

Comma-separated list of the test IDs for all active tests. This is a CSV (*A_ARG_TYPE_TestID*) list.

A test is active if it has been successfully requested and has not yet completed or been canceled. Therefore, a test ID MUST be added to *ActiveTestIDs* when its test is successfully requested, and it MUST be removed from *ActiveTestIDs* when its test completes, whether successfully or unsuccessfully, or is canceled.

See section 2.3.13 for a general description of test IDs. See Figure 2-2 for the test state transition diagram.

2.3.5. LogURIs

A comma-separated list of the URIs of the currently supported logs. This is a CSV (*A_ARG_TYPE_LogURI*).

All the URIs in the list MUST be different and MUST reference different logs, i.e. the URI is a unique key for a conceptual table of logs.

See section 2.3.33 for a description of log URIs.

2.3.6. A_ARG_TYPE_Boolean

A boolean argument.

2.3.7. A_ARG_TYPE_String

A string argument.

2.3.8. A_ARG_TYPE_UShort

An unsigned short (ui2) argument.

2.3.9. A_ARG_TYPE_UInt

An unsigned int (ui4) argument.

2.3.10. A_ARG_TYPE_DateTime

A date and time with optional time zone, plus additional conventions that apply when absolute time is not available or when the date and time are unknown.

If absolute time is not available, this SHOULD indicate the relative time since the most recent *Parent Device* restart, where the restart time is assumed to be the beginning of the first day of January of year 1, or 0001-01-01T00:00:00. For example, 2 days, 3 hours, 4 minutes and 5 seconds since restart would be expressed as 0001-01-03T03:04:05. Any value with a year value less than 1000 MUST be interpreted as a relative time since restart.

If the date and time are unknown, the following value, representing "Unknown Time", MUST be used: 0001-01-01T00:00:00.

2.3.11. A_ARG_TYPE_MSecs

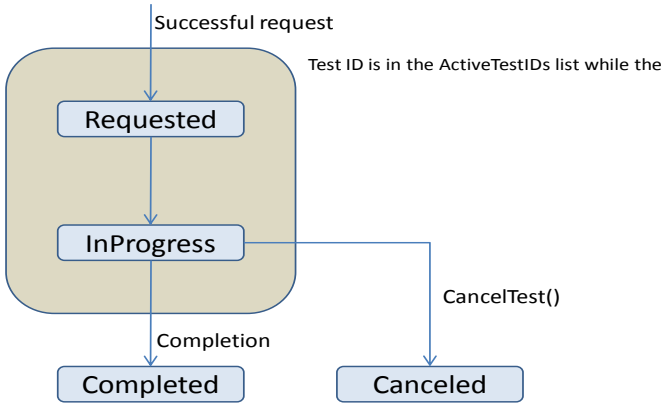
A time interval measured in milliseconds.

2.3.12. A_ARG_TYPE_RebootStatus

An output argument that indicates, for a successful *Reboot()* request, whether the *Parent Device* will be rebooting now or later. Allowed values are listed in Table 2-2.

Tab e 2-2: a edVa eL f *A_ARG_TYPE_RebootStatus*

Va e	Re . O .	De c
RebootNow	*R*	The *Reboot()* request was accepted, nothing in the current *Parent Device* state precludes an immediate reboot, and the *Parent Device* will initiate the reboot immediately.



2.3.21. A_ARG_TYPE_Hosts

A comma-separated list of IPv4/IPv6 addresses, DNS names or empty strings. This is a CSV (*A_ARG_TYPE_Host*).

2.3.22. A_ARG_TYPE_HostName

A DNS name (cannot be an IP address). It MUST NOT be an empty string.

2.3.23. A_ARG_TYPE_PingStatus

Indicates whether a ping test succeeded or, if it was not possible to complete the test, the reason for its failure. Allowed values are listed in Table 2-6.

Table 2-6: Allowed Value List for A_ARG_TYPE_PingStatus

Value	Req./Opt.	Description
Success	R	
Error_CannotResolveHostName	R	
Error_Internal	O	
Error_Other	O	
Vendor-defined	X[1]	

[1] Any vendor-defined values MUST indicate failure, i.e. they will define additional error conditions. Only *Success* indicates success.

2.3.24. A_ARG_TYPE_NSLookupStatus

Indicates whether a DNS lookup test succeeded or, if it was not possible to complete the test, the reason for its failure. Allowed values are listed in Table 2-7.

Note that *Error_DNSServerNotResolved* indicates that the DNS server was specified by name and could not be looked up, so the test could not be performed at all. Contrast this with *A_ARG_TYPE_NSLookup-Result* , which indicates the success or failure of a given repetition of the test.

Table 2-7: Allowed Value List for A_ARG_TYPE_NSLookupStatus

Value	Req./Opt.	Description
Success	R	
Error_DNSServerNotResolved	R	
Error_Internal	O	
Error_Other	O	
Vendor-defined	X[1]	

[1] Any vendor-defined values MUST indicate failure, i.e. they will define additional error conditions. Only *Success* indicates success.

2.3.25. A_ARG_TYPE_NSLookupResult

An XML document containing the result of a DNS lookup test. The normative XML Schema definition (section 4) is illustrated below. This is followed by an example XML document.

Note that *BasicManagement:1* defined just for *A_ARG_TYPE_NSLookupResult*. In *BasicManagement:2* this was incorporated into . For backwards compatibility (see section

2.5.9), has to be used when a control point invokes a *BasicManagement:1* action to return the result of a DNS lookup test.

bms:NSLookupResult **schemas-upnp-org** *dm bms*

schemas-upnp-org *dm bms*

Result *Result*

Status
Success, *Error_DNSServerNotAvailable* *Error_HostNameNot-Resolved*, *Error_Timeout* or *Error_Other*. *Status*

AnswerType
None *Authoritative* *NonAuthoritative* *AnswerType*

HostNameReturned

HostNameReturned

IPAddresses

IPAddresses

DNSServerIP
DNSServerIP

ResponseTime
ResponseTime

Result

Result *Result* *Result*
bms:NSLookupResult

This is an example XML document:

2.3.26. A_ARG_TYPE_TracerouteStatus

Indicates whether a trace-route test succeeded or, if it was not possible to complete the test, the reason for its failure. Allowed values are listed in Table 2-8.

Table 2-8: Allowed Values for A_ARG_TYPE_TracerouteStatus

Value	Req.	Opt.	Description
Success	R		
Error_CannotResolveHostName	R		
Error_MaxHopCountExceeded	R		
Error_Internal	O		
Error_Other	O		
Vendor-defined	X[1]		

[1] Any vendor-defined values MUST indicate failure, i.e. they will define additional error conditions. Only Success indicates success.

2.3.27. A_ARG_TYPE_BandwidthTestInfo

An XML document containing information about the supported bandwidth tests and about current bandwidth test server and client instances.

Note that the information about the supported bandwidth tests is almost certainly constant, but bandwidth test server and client instances could be created or deleted at any time. This information is provided because a control point might want to make a decision based on which instances currently exist, e.g. if a server is currently listening on a given port, the control point might choose to use the same port.

The *BandwidthTestInfo*, *BandwidthTestSpec* and *BandwidthTestResult* XML documents share various concepts. Section 2.6 defines and describes these common concepts. For example:

- *BandwidthTestInfo* indicates the ports on which a server can listen, whereas *BandwidthTestSpec* specifies the port on which a server will listen.
- *BandwidthTestInfo* indicates that a client can calculate jitter, whereas *BandwidthTestResult* returns the calculated jitter value.

Section 2.6 also defines several bandwidth test profiles. For example:

- For each protocol, a mandatory Baseline profile defines requirements that have to be met by all device implementations.
- For the HTTP, FTP, Echo and EchoPlus protocols, a BBF (Broadband Forum) profile defines requirements corresponding to the Broadband Forum [TR-143] tests.

The normative XML Schema definition (section 4) is described below. This is followed by some example XML documents.

bms:BandwidthTestInfo schemas-upnp-org dm bms

schemas-upnp-org dm bms

Protocol

Name

Version Profile Protocol

Name Version Profile

Protocol Name HTTP

FTP Iperf Echo EchoPlus

Version

Profile Baseline BBF

Baseline

BaseProfiles

Profile

Profile

Server

Interface

AllowedValueList

AllowedValue AllowedValue

AllowedValue AllowedValue

AllowedValue

Interface

Transport
Host
Port
TLSPort
SockBuffBytes
AppBuffBytes
Direction

TimeMSecs
LengthBytes
Username
Password
Path
BandwidthBytesPerSec
DSCP
EthernetPriority
IntervalMSecs
Threads
TCPNoDelay
TCPMSS
ExitWhenDone

File
Path

Path

RealFile Path
1 RealFile

SizeBytes
SizeBytes

MIMEType
MIMEType
File

File File File

TCPSYNTime
TCPSYNACKTime
TCPLateACKs
ROMTime
BOMTime
EOMTime
TestBytesSent
TestBytesReceived
TotalBytesSent
TotalBytesReceived

TestPacketsSent
TestPacketsReceived
TotalPacketsSent
TotalPacketsReceived
DuplicatePackets
LostPackets
OutOfSequencePackets
ResentPackets
LatencyUSecs
JitterUSecs

Instance

BandwidthTestSpec *Transport*
Port *Direction* *Path*

Instance
Instance *Instance* *Instance*
Server

Client

Client
Protocol

Protocol *Protocol* *Protocol*
bms:BandwidthTestInfo

The following example document indicates that an Iperf v2.0.5 server supports the mandatory *Baseline* profile that is defined in section 2.6.6.3. There is currently a single Iperf server instance, which is listening to TCP port 5001 on IP address 192.168.1.1.

The following example XML document indicates that an HTTP download client supports the *BBF* profile that is defined in section 2.6.6.

2.3.28. *A_ARG_TYPE_BandwidthTestSpec*

An XML document that specifies a bandwidth test.

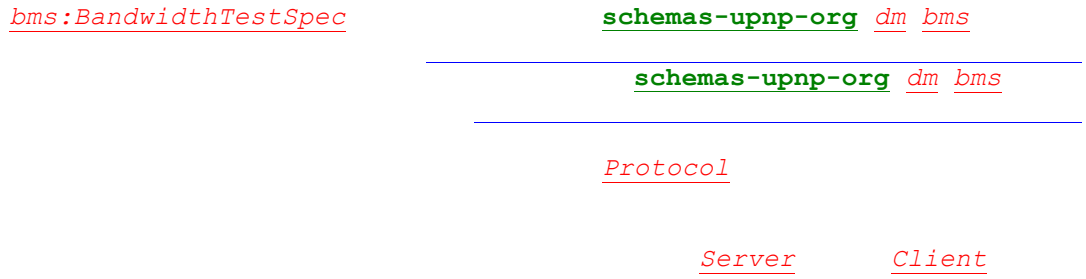
The *BandwidthTestInfo*, *BandwidthTestSpec* and *BandwidthTestResult* XML documents share various concepts. Section 2.6 defines and describes these common concepts. For example:

- *BandwidthTestInfo* indicates the ports on which a server can listen, whereas *BandwidthTestSpec* specifies the port on which a server will listen.
- *BandwidthTestInfo* indicates that a client can calculate jitter, whereas *BandwidthTestResult* returns the calculated jitter value.

Section 2.6 also defines several bandwidth test profiles. For example:

- For each protocol, a mandatory *Baseline* profile defines requirements that have to be met by all device implementations.
- For the *HTTP*, *FTP*, *Echo* and *EchoPlus* protocols, a *BBF* (Broadband Forum) profile defines requirements corresponding to the Broadband Forum [TR-143] tests.

The normative XML Schema definition (section 4) is described below. This is followed by some example XML documents.



Protocol *Name* *HTTP*
FTP *Iperf* *Echo* *EchoPlus*

Version

Profile *Baseline* *BBF*

BaseProfiles

Interface
Transport
Host
Port
TLSPort
SockBuffBytes
AppBuffBytes
Direction
TimeMSecs
LengthBytes
Username
Password
Path
BandwidthBytesPerSec
DSCP
EthernetPriority
IntervalMSecs
Threads
TCPNoDelay
TCPMSS
ExitWhenDone

Interface
Transport
Host
Port
TLSPort
SockBuffBytes
AppBuffBytes
Direction
TimeMSecs
LengthBytes
Username
Password
BandwidthBytesPerSec
DSCP
EthernetPriority
IntervalMSecs
Threads
TCPNoDelay
TCPMSS
ExitWhenDone

Server

Server

Server
Server

Client

Client
Protocol
bms:BandwidthTestSpec

The following example document describes an HTTP/1.1 Get (download) test. The profile is BBF, so additional Broadband Forum [TR-143] requirements apply.

The following more complicated example XML document describes a bidirectional Iperf test. It is

2.3.29. A_ARG_TYPE_BandwidthTestStatus

Indicates whether a bandwidth test succeeded or, if it was not possible to complete the test, the reason for its failure. Allowed values are listed in Table 2-9.

Table 2-9: Allowed Value List for A_ARG_TYPE_BandwidthTestStatus

Value	Req. / Opt.	Description
Success	R	
Error_InitConnectionFailed	R	
Error_NoResponse	R	
Error_TransferFailed	R	
Error_PasswordRequestFailed	R	
Error_LoginFailed	R	
Error_NoTransferMode	R	
Error_IncorrectSize	R	
Error_NoTransferComplete	R	
Error_Timeout	R	
Error_Other	R	
Error_FTPNoPASV	R	
Error_FTPNoCWD	R	
Error_FTPNoSTOR	R	
Vendor-defined	X[1]	

[1] Any vendor-defined values MUST indicate failure, i.e. they will define additional error conditions. Only Success indicates success.

2.3.30. A_ARG_TYPE_BandwidthTestResult

An XML document that describes a bandwidth test result.

The *BandwidthTestInfo*, *BandwidthTestSpec* and *BandwidthTestResult* XML documents share various concepts. Section 2.6 defines and describes these common concepts. For example:

- *BandwidthTestInfo* indicates the ports on which a server can listen, whereas *BandwidthTestSpec* specifies the port on which a server will listen.
- *BandwidthTestInfo* indicates that a client can calculate jitter, whereas *BandwidthTestResult* returns the calculated jitter value.

Section 2.6 also defines several bandwidth test profiles. For example:

- For each protocol, a mandatory *Baseline* profile defines requirements that have to be met by all device implementations.
- For the *HTTP*, *FTP*, *Echo* and *EchoPlus* protocols, a *BBF* (Broadband Forum) profile defines requirements corresponding to the Broadband Forum [TR-143] tests.

The normative XML Schema definition (section 4) is described below. It is followed by some example XML documents.

bms:BandwidthTestResult schemas-upnp-org dm bms

schemas-upnp-org dm bms

Protocol

Protocol Name HTTP
FTP Iperf Echo EchoPlus

Version

Profile Baseline BBF

Spec

Protocol

Spec

Server Interval

Stream

TCPSYNTime
TCPSYNACKTime
TCPLateACKs

TCPSYNTime
TCPSYNACKTime
TCPLateACKs

```
        ROMTime                                  ROMTime
        BOMTime                                    BOMTime
        EOMTime                                EOMTime
        TestBytesSent                            TestBytesSent
        TestBytesReceived                                TestBytesReceived
        TotalBytesSent                           TotalBytesSent
        TotalBytesReceived                                 TotalBytesReceived
        TestPacketsSent                            TestPacketsSent
        TestPacketsReceived                                TestPacketsReceived
        TotalPacketsSent                         TotalPacketsSent
        TotalPacketsReceived                               TotalPacketsReceived
        DuplicatePackets                                        DuplicatePackets
        LostPackets                        LostPackets
        OutOfSequencePackets                             OutOfSequencePackets
        ResentPackets                          ResentPackets
        LatencyUSecs                                     LatencyUSecs
        JitterUSecs                                    JitterUSecs
      Server

    Client


      Client
    Protocol
  bms:BandwidthTestResult
```

The following example document shows the result of an HTTP/1.1 Get (download) test.

The following more complicated example XML document shows the results of an *Iperf* test with a reporting interval.

2.3.31. *A_ARG_TYPE_Interfaces*

Indicates which IP interfaces should be reset by an interface reset test. Allowed values are listed in Table 2-10

Tab e 2-10: a edVa eL f *A_ARG_TYPE_Interfaces*

Va e	Re . O .	De c
AllInterfaces	*R*	All IP interfaces
RequestInterface	*R*	The IP interface on which the action request was received is to be reset.
NorthboundInterfaces	*O*	Relevant only to an Internet Gateway Device (a router that is connected to the Internet); the IP interface or interfaces via which traffic can be sent to or received from the Internet.

Va e	Re . O .	De c
Vendor-defined	*X*[1]	MUST be the string "X_UPNP_ORG_" followed by the name by which an IP interface is known in the [CMS] data model, i.e. the value of the corresponding parameter.

[1] Any vendor-defined value MUST be the name by which an IP interface is known in the [CMS] data model.

2.3.32. *A_ARG_TYPE_InterfaceResetStatus*

Indicates whether an interface reset test succeeded or, if it was not possible to complete the test, the reason for its failure. Allowed values are listed in Table 2-11.

Tab e 2-11: a edVa eL f *A_ARG_TYPE_InterfaceResetStatus*

Va e	Re . O .	De c
Success	*R*	
Error_Other	*R*	
Vendor-defined	*X*[1]	

[1] Any vendor-defined values MUST indicate failure, i.e. they will define error conditions. Only *Success* indicates success.

2.3.33. *A_ARG_TYPE_LogURI*

A URN or URL that identifies one of the logs exposed by the *Parent Device*.

Logs that are specified by UPnP Forum working committees MUST always be identified via a URN of a specified format and meaning, e.g. "urn:upnp-org:committee:mylog".

Vendor-specific logs MAY be specified via either a URN or a URL. If a vendor-specific log is specified via a URL, the URL MUST be relative to the URL from which the *Parent Device* description was retrieved, e.g. a value of "mylog.xml" might correspond to an absolute URL of "http://192.168.1.1/logs/-mylog.xml". A relative URL is required so that it can be used across a population of devices (because it is independent of protocol, credentials, IP address and port number).

The reason that logs that are specified by UPnP Forum working committees have to be identified by a URN (rather than a URL) is that working committees can define URN formats but could not reasonably specify URLs because this would constrain the implementation.

2.3.34. *A_ARG_TYPE_LogURL*

A log URL via which one of the logs exposed by the *Parent Device* can be retrieved.

- The URL SHOULD be relative to the URL from which the *Parent Device* description was retrieved, e.g. a value of "mylog.xml" might correspond to an absolute URL of "http://192.168.1-.1/logs/mylog.xml". A relative URL is preferred because the same value can be used across multiple control interfaces, e.g. an IPv4 and an IPv6 interface.
- The URL MUST NOT include a "userinfo" component, as defined in [URI]. This is to avoid conflict with any log security access mechanism.

2.3.35. A_ARG_TYPE_LogLevel

Indicates the lowest logging level that is included in a given log. Allowed values are listed in Table 2-12. These values are in decreasing order of severity (most severe first, least severe last). When the logging level is set to a given value, items at that level and all higher severity levels (i.e. levels defined earlier in the table) are logged. For example, if it is set to *Error*, items at level *Error*, *Critical*, *Alert* and *Emergency* will be logged.

Logs can contain lists of UPnP actions invoked on the *Parent Device*. The implementation MAY choose to use different logging levels to record actions that do and do not affect the *Parent Device* state, e.g. *Notice* for actions that affect state and *Informational* for actions that do not affect state.

Tab e 2-12: a edVa eL f A_ARG_TYPE_LogLevel[1]

Va e	Re . O . De c
Emergency	*R*
Alert	*R*
Critical	*R*
Error	*R*
Warning	*R*
Notice	*R*
Informational	*R*
Debug	*R*
Vendor-defined	*X*[2]

[1] These are the syslog message severities that are defined in [SYSLOG] section 4.1.1.

[2] Because log levels are ordered, the definitions of any vendor-defined values MUST indicate where they are to be inserted in the list of log levels.

2.3.36. A_ARG_TYPE_LogMaxSize

Indicates the maximum possible size (in bytes) of a given log. A value of zero indicates that the maximum possible size is not fixed or is not known.

2.3.37. A_ARG_TYPE_ACL

An XML document containing the *BasicManagement:2* access control list (ACL). This is a list of zero or more action ACL entries, each of which specifies a filter expression and a set of associated *Roles*. See section 2.2.4.1 for an explanation of how the ACL is used.

Note that ACL entries are only provided for actions that have non-empty *Restricted Role Lists*. For such actions, the ACL filter expressions determine, for control points that possess a given *Role*, which action argument values will permit the action to be invoked.

The normative XML Schema definition is illustrated below. This is followed by an example XML document. The XML Schema does not define the filter expression syntax, which is described below the example.

```
bms:ACL                    schemas-upnp-org dm bms

                           schemas-upnp-org dm bms
```

ACLEntry

Action *Filter*

ACLEntry *ACLEntry*

Action

Action

Filter

Filter

Roles

Roles

ACLEntry

ACLEntry *ACLEntry* *ACLEntry*

bms:ACL

This example illustrates two ACL entries, which allow control points that possess the *Basic* *Role* to invoke the *InterfaceReset()* action if the *Interfaces* argument has the value "*RequestInterface*", and to invoke the *SetLogInfo()* action if the *LogURI* argument contains the string " ".

The filter expression syntax is based on XPath 1.0 [XPATH-1.0]. All the lexical rules, operator precedence, type conversions etc are as specified by XPath, but the concept of "path" is replaced with the much simpler concept of "action argument".

The following EBNF-style syntax [EBNF] MUST be supported:

```
1.  Filter              ::=  Expr

2.  Expr                ::=  EqualityExpr
                             | FunctionCall

3.  EqualityExpr        ::=  | ArgumentName "=" Constant
                             | ArgumentName "!=" Constant

4.  Constant            ::=  Literal (string in double quotes)
                             | Number (signed or unsigned integer)
```

5. `FunctionCall ::= “contains(” ArgumentName “,” Literal “)”`

This is sufficient to support the requirements of the current version of this specification, but it is possible that a vendor will need to support more complicated filters. In this case, the filter expression syntax can be extended as follows:

- Additional operators MAY be supported by extending the supported XPath 1.0 subset.
- All extensions MUST use XPath 1.0 lexical rules, operator precedence, type conversions etc.
- Vendor-specific function names MUST be namespace-qualified.
- Vendor-specific namespaces MUST be declared using the usual XML mechanisms.
- Vendor-specific functions SHOULD be defined only where no standard XPath 1.0 function provides a reasonable alternative. This is to maximize the chance that a control point will be able to parse and understand such a vendor extension.

For example, a vendor who needed a filter to be able to reference more than one argument, check for argument values beginning with a specified substring, and call a vendor-specific function might support the following ACL:

A control point that understood XPath 1.0 expression syntax would be able to parse the filter expression but would have no way of knowing what the function did.

2.4. E e g a d M de a

Tab e 2-13: E e M de a

Va ab e Na e	E e ed	M de a ed E e	Ma E e Ra e[1]	L g ca C b a	M De a e E e [2]
DeviceStatus	*Yes*	*Yes*	1.0 second		
SequenceMode	*Yes*	*No*			
TestIDs	*Yes*	*Yes*	0.2 seconds		
ActiveTestIDs	*Yes*	*Yes*	0.2 seconds		
LogURIs	*Yes*	*No*			
Non-standard state variables implemented by an UPnP vendor go here.	*TBD*	*TBD*	*TBD*	*TBD*	*TBD*

[1] Determined by N, where Rate = (Event)/(N secs).
[2] (N) * (allowedValueRange Step).

2.4.1. SSDP A ce e Mecha

In addition to [UDA1.0] GENA eventing, a *Parent Device* can use a new *Announcement.dm.upnp.org* SSDP header to announce important *Parent Device* state information to control points and other UPnP controlled devices. The new header takes the following form, where *field-value* identifies the state information (Table 2-14).

Announcement.dm.upnp.org: *field-value*

Tab e 2-14: A ed Va e f *Announcement.dm.upnp.org field-value*

Va e	Re . O .	De c
AboutToReboot	*R*	Indicates that the *Parent Device* and/or the targeted Execution Environment and/or the Operating System are about to reboot. The corresponding internal state can be set to "_" for various reasons, including autonomous reboots and *Reboot()* requests.
AboutToBaselineReset	*R*	Indicates that the *Parent Device* and/or the targeted Execution Environment and/or the Operating System are about to return to their baseline reset state. The corresponding internal state can be set to "_" for various reasons, including autonomous baseline resets and *BaselineReset()* requests.
Vendor-defined	*X*[1]	

[1] Any vendor-defined values MUST obey the usual naming rules for vendor extensions, as defined in [UDA1.0].

The usual [HTTP] header rules apply, e.g. with regard to case-dependence (the header name is case-independent and *field-value* is case-dependent), quoting (*field-value* can be quoted) and provision of multiple values (two headers with *field-values* of *A* and *B* are equivalent to a single header with a *field-value* of *A*,*B*).

The following additional requirements relate to the *Announcement.dm.upnp.org* mechanism:

- The mechanism MUST only be used to announce important *Parent Device* state information that cannot reasonably be sent using GENA eventing.
- Each *field-value* corresponds to a piece of Boolean internal state information. It MUST be included in every *Parent Device* SSDP message (and SHOULD be included in every SSDP message for devices / services embedded within the *Parent Device*) if and only if the corresponding internal state is "_" when the SSDP message is sent.
- The above requirement SHOULD apply to every SSDP message for devices / services embedded within the *Parent Device*. This is an application of the requirement, in section 1.1, that actions or other operations on a *Parent Device* should apply to all levels of its sub-tree.

Note that it is unlikely that either the *AboutToReboot* or the *AboutToBaselineReset* internal state will be "_" when _____ messages are sent, so *Announcement.dm.upnp.org: AboutToReboot* and *Announcement.dm.upnp.org: AboutToBaselineReset* headers are likely to be present only in _____ messages. However, the use of the *Announcement.dm.upnp.org* mechanism with _____ messages is not specifically forbidden.

2.5. Ac

Table 2-15 lists actions, their device and control point support requirements, and their recommended *Role Lists* and *Restricted Role Lists*. Only the standard *DeviceProtection:1* *Admin*, *Basic* and *Public* *Roles* are mentioned, because the device manufacturer is free to choose how the *dm:ThirdPartyAdmin* and *dm:UserAdmin* *Roles* (defined in [DPS]) relate to the *Admin* and *Basic* *Roles*, and it would therefore be impossible to include them in the table.

Section 2.2.4 defined *Non-Restrictable* and *Restrictable* actions and pointed out that all *Non-Restrictable* actions have a *Role List* of "*Public*" and an empty *Restricted Role List*. Table 2-15 explicitly indicates which actions are *Non-Restrictable*.

Tab e 2-15: Ac

Na e	De ce R/O[1]	C P R/O	Rec e ded R e L[2]	Rec e ded Re c ed R e L[3]
Reboot()	*O*	*O*	*Admin*	
BaselineReset()	*O*	*O*	*Admin*	
GetDeviceStatus()	*R*	*O*	*Public*[4]	
SetSequenceMode()	*O*	*O*	*Admin* *Basic*	*Public*
GetSequenceMode()	*O*	*O*	*Public*[4]	
Ping()	*CR*[5]	*CR*[5]	*Public*[4]	
GetPingResult()	*CR*[5]	*CR*[5]	*Public*[4]	
NSLookup()	*CR*[6]	*CR*[6]	*Public*[4]	
GetNSLookupResult()	*CR*[6]	*CR*[6]	*Public*[4]	
Traceroute()	*CR*[7]	*CR*[7]	*Public*[4]	
GetTracerouteResult()	*CR*[7]	*CR*[7]	*Public*[4]	
GetBandwidthTestInfo()				

Na e	De ce R/O[1]	C P R/O	Rec e ded R e L [2]	Rec e ded Re c ed R e L [3]
GetTestInfo()	*CR*[11]	*CR*[11]	*Public*[4]	
CancelTest()	*CR*[11]	*CR*[11]	*Admin*	*Basic* *Public*
GetLogURIs()	*O*	*O*	*Public*[4]	
SetLogInfo()	*O*	*O*	*Admin*	*Basic*
GetLogInfo()	*O*	*O*	*Public*[4]	
GetACLData()	*CR*[12]	*CR*[12]	*Admin* *Basic*	*Public*
Non-standard actions implemented by an UPnP vendor go here.	*X*	*X*		

[1] *R* = REQUIRED, *O* = OPTIONAL, *CR* = CONDITIONALLY REQUIRED, *X* = Non-standard.

[2] The *Role List* contains *Roles* that are authorized to invoke the corresponding action in all contexts. For *Restrictable* actions, the device manufacturer can choose different values for the *Role List*.

[3] The *Restricted Role List* contains *Roles* that are authorized to invoke the corresponding action only in certain contexts. See the individual action definitions for details. For *Restrictable* actions, the device manufacturer can choose different values for the *Restricted Role List*.

[4] This action is *Non-Restrictable*. For *Non-Restrictable* actions, the *Role List* MUST be "*Public*" and the *Restricted Role List* MUST be empty, i.e. the device manufacturer can not choose different values for the *Role List* or for the *Restricted Role List*.

[5] REQUIRED if the *Ping Diagnostics Feature* is supported.

[6] REQUIRED if the *NSLookup Diagnostics Feature* is supported.

[7] REQUIRED if the *Traceroute Diagnostics Feature* is supported.

[8] REQUIRED if the *BandwidthTest Diagnostics Feature* is supported.

[9] REQUIRED if the *InterfaceReset Diagnostics Feature* is supported.

[10] REQUIRED if the *SelfTest Diagnostics Feature* is supported.

[11] REQUIRED if any of the *Diagnostics Features* are supported, e.g. the baseline *Diagnostics Feature* or the *Ping Diagnostics Feature*.

[12] REQUIRED if the *Security Feature* is supported.

2.5.1. *Reboot()*

The *Reboot()* action reboots the *Parent Device* and possibly (see section 2.5.1.2) the targeted Execution Environment and/or the Operating System. The *Parent Device* SHOULD send out the appropriate _____ messages before rebooting.

The *Parent Device* might be doing something, e.g. providing a service, that means that an immediate reboot is not desirable. The implementation MUST NOT reject a *Reboot()* request for this reason, but can choose to defer the reboot and to indicate this via a *RebootStatus* value of *RebootLater*.

2.5.1.1. Arguments

Tab e 2-16: A g e f *Reboot()*

A g e	D ec	e a edS a eVa ab e
RebootStatus	*OUT*	*A_ARG_TYPE_RebootStatus*

2.5.1.2. Device Requirements

As specified by *DeviceProtection:1*, any control point that possesses any of the *Roles* in the action's *Role List* MUST be permitted to invoke this action regardless of the values of any action input arguments. If the *Security Feature* is not supported, all actions are permitted, i.e. behavior is the same as if the action had a *Role List* of "*Public*".

Otherwise, if all of the following conditions are met, any control point that possesses any of the *Roles* in the action's *Restricted Role List* MUST be permitted to invoke the action:

- The action was invoked over a TLS connection.
- The control point *Identity* is present in the *DeviceProtection:1* ACL.

2.5.1.3. Dependency on State

The *Parent Device* MAY indicate, via the following [CMS] parameters, whether the *Reboot()* action will reboot the targeted Execution Environment and/or the Operating System:

- If the [CMS] parameter is present and has the value " ", the *Reboot()* action MUST reboot the targeted Execution Environment. If [CMS] is not supported, the parameter is absent, or it has the value " ", control points cannot determine whether or not the *Reboot()* action will reboot the targeted Execution Environment.

- If the [CMS] parameter is present and has the value " ", the *Reboot()* action MUST reboot the Operating System. If [CMS] is not supported, the parameter is absent, or it has the value " ", control points cannot determine whether or not the *Reboot()* action will reboot the Operating System.

2.5.1.4. Effect on State

On successful completion of a *Reboot()* request, the *Parent Device* *AboutToReboot* internal state is set to "_" (section 2.4.1), meaning that each _____ message will include an *Announcement.dm.upnp.org: -AboutToReboot* header.

Once the *Parent Device* *AboutToReboot* internal state has been set to "_":

- If *RebootStatus* is *RebootNow*, the *Parent Device* MUST immediately initiate the reboot procedure.

- If *RebootStatus* is *RebootLater*, the *Parent Device* MUST initiate the reboot procedure as soon as it can, consistent with its responsibility to provide normal service.

- On reboot, all *Reboot()* requests are considered to have been satisfied, i.e. *Reboot()* requests don't stack up.

Any action requests received after the successful completion of a *Reboot()* request but before the reboot has occurred SHOULD be rejected with a 501 (Action Failed) error code. This requirement applies to any of the *Parent Device's* services, including any services within its embedded devices. This is an application of the requirement, in section 1.1, that actions or other operations on a *Parent Device* should apply to all levels of its sub-tree.

2.5.1.5. Errors

Tab e 2-17: E C de f *Reboot()*

e C de	e De c	De c
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
606	Action not authorized	The action requested requires authorization and the sender was not authorized.

2.5.2. *BaselineReset()*

The *BaselineReset()* action returns the *Parent Device*, and possibly (see section 2.5.2.2) the targeted Execution Environment and/or the Operating System, to their baseline states. The *Parent Device* SHOULD send out the appropriate _____ messages before restoring the baseline settings.

Note that the action is called *BaselineReset()* rather than the more common *FactoryReset()*. This is to emphasize that the baseline state does not need to be the factory state. For example, the baseline state might use a stable version of the firmware that is more recent that the factory firmware.

BaselineReset() SHOULD apply to all devices / services embedded within the *Parent Device*. This is an application of the requirement, in section 1.1, that actions or other operations on a *Parent Device* should apply to all levels of its sub-tree.

2.5.2.1. Arguments

None.

2.5.2.2. Device Requirements

As specified by *DeviceProtection:1*, any control point that possesses any of the *Roles* in the action's *Role List* MUST be permitted to invoke this action regardless of the values of any action input arguments. If the *Security Feature* is not supported, all actions are permitted, i.e. behavior is the same as if the action had a *Role List* of "*Public*".

Otherwise, if all of the following conditions are met, any control point that possesses any of the *Roles* in the action's *Restricted Role List* MUST be permitted to invoke the action:

- The action was invoked over a TLS connection.
- The control point *Identity* is present in the *DeviceProtection:1* ACL.

2.5.2.3. Dependency on State

The *Parent Device* MAY indicate, via the following [CMS] parameters, whether the *BaselineReset()* action will return the targeted Execution Environment and/or the Operating System to their baseline states:

- If the
  [CMS] parameter is present and has the value " ", the *BaselineReset()* action MUST return the targeted Execution Environment to its baseline state. If [CMS] is not supported, the parameter is

absent, or it has the value " ", control points cannot determine whether or not the *BaselineReset()* action will return the targeted Execution Environment to its baseline state.

- If the [CMS] parameter is present and has the value " ", the *BaselineReset()* action MUST return the Operating System to its baseline state. If [CMS] is not supported, the parameter is absent, or it has the value " ", control points cannot determine whether or not the *BaselineReset()* action will return the Operating System to its baseline state.

2.5.2.4. Effect on State

On successful completion of a *BaselineReset()* request, the *Parent Device* *AboutToBaselineReset* internal state is set to "_" (section 2.4.1), meaning that each _____ message will include an *Announcement.dm.upnp.org: AboutToBaselineReset* header.

Any action requests received after the successful completion of a *BaselineReset()* request but before the baseline reset has occurred SHOULD be rejected with a 501 (Action Failed) error code. This requirement applies to any of the *Parent Device*'s services, including any services within its embedded devices. This is an application of the requirement, in section 1.1, that actions or other operations on a *Parent Device* should apply to all levels of its sub-tree.

2.5.2.5. Errors

Tab e 2-18: E C de f *BaselineReset()*

e C de e De c	De c
400-499	See UPnP Device Architecture section on Control.
500-599	See UPnP Device Architecture section on Control.
600-699	See UPnP Device Architecture section on Control.
606	Action not authorized	The action requested requires authorization and the sender was not authorized.

2.5.3. *GetDeviceStatus()*

The *GetDeviceStatus()* action returns the current value of the *DeviceStatus* state variable.

2.5.3.1. Arguments

Tab e 2-19: A g e f *GetDeviceStatus()*

A g e	D ec	e a edS a eVa ab e
DeviceStatus	*OUT*	*DeviceStatus*

2.5.3.2. Device Requirements

This action returns the value of an evented state variable. This value is freely available to all control points, so the action is *Non-Restrictable* and all control points MUST be permitted to invoke the action regardless of which *Roles* they possess.

2.5.3.3. Dependency on State

None.

2.5.3.4. Effect on State

None.

2.5.3.5. Errors

Table 2-20: Error Codes for GetDeviceStatus()

Error Code	Error Description	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.

2.5.4. SetSequenceMode()

The *SetSequenceMode()* action sets the value of the *SequenceMode* state variable. The arguments are used as follows:

- : is the new value of the *SequenceMode* state variable.
- : returns the previous value of the *SequenceMode* state variable, so a control point that sets *SequenceMode* to "_" will know whether another control point had already set it to "_".

2.5.4.1. Arguments

Table 2-21: Argument for SetSequenceMode()

Argument	Direction	relatedStateVariable
NewSequenceMode	*IN*	*SequenceMode*
OldSequenceMode	*OUT*	*SequenceMode*

2.5.4.2. Device Requirements

As specified by *DeviceProtection:1*, any control point that possesses any of the *Roles* in the action's *Role List* MUST be permitted to invoke this action regardless of the values of any action input arguments. If the *Security Feature* is not supported, all actions are permitted, i.e. behavior is the same as if the action had a *Role List* of "*Public*".

Otherwise, if all of the following conditions are met, any control point that possesses any of the *Roles* in the action's *Restricted Role List* MUST be permitted to invoke the action:

- The action was invoked over a TLS connection.
- The control point *Identity* is present in the *DeviceProtection:1* ACL.

2.5.4.3. Dependency on State

None.

2.5.4.4. Effect on State

The *SequenceMode* state variable is set to the requested value.

2.5.4.5. Errors

Table 2-22: Error Codes for SetSequenceMode()

Error Code	Error Description	Description
400-499	TBD	See UPnP Device Architecture section on Control.

e C de	e De c	De c
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
606	Action not authorized	The action requested requires authorization and the sender was not authorized.

2.5.5. GetSequenceMode()

The *GetSequenceMode()* action returns the current value of the *SequenceMode* state variable.

2.5.5.1. Arguments

Tab e 2-23: A g e f GetSequenceMode()

A g e	D ec	e a edS a eVa ab e
SequenceMode	*OUT*	*SequenceMode*

2.5.5.2. Device Requirements

This action returns the value of an evented state variable. This value is freely available to all control points, so the action is *Non-Restrictable* and all control points MUST be permitted to invoke the action regardless of which *Roles* they possess.

2.5.5.3. Dependency on State

None.

2.5.5.4. Effect on State

None.

2.5.5.5. Errors

Tab e 2-24: E C de f GetSequenceMode()

e C de	e De c	De c
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.

2.5.6. Ping()

The *Ping()* action requests an IP-layer ping test. If a ping test is already active, the service MAY reject the request.

The ping test involves sending ICMP echo packets to the specified host, as specified in [ICMP]. The input arguments are used as follows:

- : is the name or address of the ICMP echo packet destination. It MUST NOT be an empty string.
- : is the number of packets to send. A value of zero requests use of an implementation-chosen default number of repetitions.
- : is the length of time (in milliseconds) to wait between sending each packet (regardless of whether a response has been received to the previous packet). A value of zero requests use of an implementation-chosen interval. Note that this argument is misnamed: it is not really a timeout.

- : is the size of each packet's data block (the data block's contents are implementation-specific). A value of zero requests use of an implementation-chosen default data block size.
- : is the DiffServ Code Point [DSCP] value in each packet's IP header. A value of zero implies default (best effort) treatment.

2.5.6.1. Arguments

Tab e 2-25: A g e f *Ping()*

A g e	D ec	e a edS a eVa ab e
Host	*IN*	*A_ARG_TYPE_Host*
NumberOfRepetitions	*IN*	*A_ARG_TYPE_UInt*
Timeout	*IN*	*A_ARG_TYPE_MSecs*
DataBlockSize	*IN*	*A_ARG_TYPE_UShort*
DSCP	*IN*	*A_ARG_TYPE_DSCP*
TestID	*OUT*	*A_ARG_TYPE_TestID*

2.5.6.2. Device Requirements

This action is *Non-Restrictable* and all control points MUST be permitted to invoke the action regardless of which *Roles* they possess.

The implementation MAY forbid use of a of less than 1000 (1 second) by unauthorized control points.

2.5.6.3. Dependency on State

None.

2.5.6.4. Effect on State

When a ping test is successfully requested, the *TestID* MUST be added to the *TestID* and *ActiveTestIDs* state variables.

2.5.6.5. Errors

Tab e 2-26: E C de f *Ping()*

e C de	e De c	De c
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
703	Test Already Active	A test of this type is already active (and the implementation doesn't support multiple active instances of a given test type).
704	Capabilities Preclude Test	Test arguments are individually valid but, taken together, describe a test that is beyond the service's capabilities.
705	State Precludes Test	Service state precludes performing this test.

2.5.7. GetPingResult()

The GetPingResult() action returns the results of a completed IP-layer ping test. The output arguments are defined as follows:

- : indicates the overall success or failure of the test (if the test failed, the values of the remaining output arguments – other than – are not specified, and MUST be ignored).
- : a free-format string that can contain additional information about the test result.
- : is the number of successful pings (those for which a successful response was received prior to the timeout).
- : is the number of failed pings (SuccessCount + FailureCount MUST equal NumberOf-Repetitions).
- : is the average response time (in milliseconds) over all successful pings, or zero if there were none.
- : is the minimum response time (in milliseconds) over all successful pings, or zero if there were none.
- : is the maximum response time (in milliseconds) over all successful pings, or zero if there were none.

2.5.7.1. Arguments

Tab e 2-27: A g e f GetPingResult()

A g e	D ec	e a edS a eVa ab e
TestID	IN	A_ARG_TYPE_TestID
Status	OUT	A_ARG_TYPE_PingStatus
AdditionalInfo	OUT	A_ARG_TYPE_String
SuccessCount	OUT	A_ARG_TYPE_UInt
FailureCount	OUT	A_ARG_TYPE_UInt
AverageResponseTime	OUT	A_ARG_TYPE_MSecs
MinimumResponseTime	OUT	A_ARG_TYPE_MSecs
MaximumResponseTime	OUT	A_ARG_TYPE_MSecs

2.5.7.2. Device Requirements

This action is *Non-Restrictable* and all control points MUST be permitted to invoke the action regardless of which *Roles* they possess.

2.5.7.3. Dependency on State

A test with the specified TestID needs previously to have been successfully requested, and to have completed.

2.5.7.4. Effect on State

None.

2.5.7.5. Errors

Tab e 2-28: E C de f GetPingResult()

e C de	e De c	De c
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
706	No Such Test	No test with the specified TestID was found.
707	Wrong Test Type	TestID is valid but refers to a different test type.
708	Invalid Test State	The TestID is valid but test results are not available.

2.5.8. *NSLookup()*

The *NSLookup()* action requests an IP-layer DNS lookup. If a lookup test is already active, the service MAY reject the request.

The lookup test involves contacting and querying a DNS server as specified in [DNS]. The input arguments are used as follows:

- : is the name of the host to look up. The current domain name MUST be used unless the name is a fully qualified name.
- : is the name or address of the DNS server. The name of this server will be resolved using the default DNS server unless an IP address is provided. If an empty string is specified, the default DNS server will be used.
- : is the number of lookups to perform. If a lookup fails the test MAY be terminated without completing the full number of repetitions. A value of zero requests use of an implementation-chosen default number of repetitions.
- : is the length of time (in milliseconds) to wait for each response before sending the next request. A value of zero requests use of an implementation-chosen timeout.

2.5.8.1. Arguments

Tab e 2-29: A g e f *NSLookup()*

A g e	D ec	e a edS a eVa ab e
HostName	*IN*	*A_ARG_TYPE_HostName*
DNSServer	*IN*	*A_ARG_TYPE_Host*
NumberOfRepetitions	*IN*	*A_ARG_TYPE_UInt*
Timeout	*IN*	*A_ARG_TYPE_MSecs*
TestID	*OUT*	*A_ARG_TYPE_TestID*

2.5.8.2. Device Requirements

This action is *Non-Restrictable* and all control points MUST be permitted to invoke the action regardless of which *Roles* they possess.

2.5.8.3. Dependency on State

None.

2.5.8.4. Effect on State

When a DNS lookup test is successfully requested, the *TestID* MUST be added to the *TestID* and *ActiveTestIDs* state variables.

2.5.8.5. Errors

Table 2-30: Error Codes for NSLookup()

Error Code	Error Description	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
703	Test Already Active	A test of this type is already active (and the implementation doesn't support multiple active instances of a given test type).
704	Capabilities Preclude Test	Test arguments are individually valid but, taken together, describe a test that is beyond the service's capabilities.
705	State Precludes Test	Service state precludes performing this test.

2.5.9. GetNSLookupResult()

The *GetNSLookupResult()* action returns the results of a completed IP-layer DNS lookup test. The output arguments are defined as follows:

- : indicates the overall success or failure of the test (if the test failed, the values of the remaining output arguments – other than – are not specified, and MUST be ignored).
- : a free-format string that can contain additional information about the test result.
- : is the number of successful DNS lookups (those for which a successful response was received prior to the timeout).
- : is an XML document containing the result of the DNS lookup test. See section 2.3.25 for the definition of and examples of the test result.

2.5.9.1. Arguments

Table 2-31: Args for GetNSLookupResult()

Argument	Dir	relatedStateVariable
TestID	*IN*	*A_ARG_TYPE_TestID*
Status	*OUT*	*A_ARG_TYPE_NSLookup-Status*
AdditionalInfo	*OUT*	*A_ARG_TYPE_String*
SuccessCount	*OUT*	*A_ARG_TYPE_UInt*
Result	*OUT*	*A_ARG_TYPE_NSLookup-Result*

2.5.9.2. Device Requirements

This action is *Non-Restrictable* and all control points MUST be permitted to invoke the action regardless of which *Roles* they possess.

2.5.9.3. Dependency on State

A test with the specified *TestID* needs previously to have been successfully requested, and to have completed.

2.5.9.4. Effect on State

None.

2.5.9.5. Errors

Table 2-32: Error Code Effect GetNSLookupResult()

Error Code	Error Name	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
706	No Such Test	No test with the specified TestID was found.
707	Wrong Test Type	TestID is valid but refers to a different test type.
708	Invalid Test State	The TestID is valid but test results are not available.

2.5.10. Traceroute()

The *Traceroute()* action requests an IP-layer trace-route test. If a trace-route test is already active, the service MAY reject the request.

Traceroute implementations vary, but all send probe packets to the specified host, increasing the time-to-live (TTL) value from an initial value of 1, and relying on receiving ICMP time exceeded messages, as specified in [ICMP]. The input arguments are used as follows:

- : is the name or address of the host to find a route to. It MUST NOT be an empty string.
- : is the length of time (in milliseconds) to wait for each reply. A value of zero requests use of an implementation-chosen timeout.
- : is the size of each probe packet's data block (the data block's contents are implementation-specific). A value of zero requests use of an implementation-chosen default data block size.
- : is the maximum number of hops used in probe packets, i.e. the maximum time-to-live (TTL). A value of zero requests use of an implementation-chosen default maximum hop count.
- : is the DiffServ Code Point value in each probe packet's IP header. A value of zero implies default (best effort) treatment.

2.5.10.1. Arguments

Table 2-33: Arguments for *Traceroute()*

Argument	Direction	relatedStateVariable
Host	*IN*	*A_ARG_TYPE_Host*
Timeout	*IN*	*A_ARG_TYPE_MSecs*
DataBlockSize	*IN*	*A_ARG_TYPE_UShort*
MaxHopCount	*IN*	*A_ARG_TYPE_UInt*
DSCP	*IN*	*A_ARG_TYPE_DSCP*
TestID	*OUT*	*A_ARG_TYPE_TestID*

2.5.10.2. Device Requirements

This action is *Non-Restrictable* and all control points MUST be permitted to invoke the action regardless of which *Roles* they possess.

2.5.10.3. Dependency on State

None.

2.5.10.4. Effect on State

When a trace-route test is successfully requested, the *TestID* MUST be added to the *TestID* and *ActiveTestIDs* state variables.

2.5.10.5. Errors

Tab e 2-34: E C de f *Traceroute()*

e C de e De c	De c
400-499	See UPnP Device Architecture section on Control.
500-599	See UPnP Device Architecture section on Control.
600-699	See UPnP Device Architecture section on Control.
703	Test Already Active	A test of this type is already active (and the implementation doesn't support multiple active instances of a given test type).
704	Capabilities Preclude Test	Test arguments are individually valid but, taken together, describe a test that is beyond the service's capabilities.
705	State Precludes Test	Service state precludes performing this test.

2.5.11. *GetTracerouteResult()*

The *GetTracerouteResult()* action returns the results of a completed IP-layer trace-route test. The output arguments are defined as follows:

- : indicates the overall success or failure of the test (if the test failed, the values of the remaining output arguments – other than – are not specified, and MUST be ignored).
- : a free-format string that can contain additional information about the test result.
- : is the average response time (in milliseconds) for the most recent probe, i.e. for the messages that actually reached the host.
- : is a comma-separated list of the hosts along the discovered route. Each host SHOULD be an IP address (not a DNS name). If a host could not be contacted, the corresponding entry in the list is empty, i.e. there will be two consecutive commas in the list, as in "host1,,host3".

2.5.11.1. Arguments

Tab e 2-35: A g e f *GetTracerouteResult()*

A g e	D ec	e a edS a eVa ab e
TestID	*IN*	*A_ARG_TYPE_TestID*
Status	*OUT*	*A_ARG_TYPE_Traceroute-Status*
AdditionalInfo	*OUT*	*A_ARG_TYPE_String*

A g e	D ec	e a edS a eVa ab e
ResponseTime	*OUT*	*A_ARG_TYPE_MSecs*
HopHosts	*OUT*	*A_ARG_TYPE_Hosts*

2.5.11.2. Device Requirements

This action is *Non-Restrictable* and all control points MUST be permitted to invoke the action regardless of which *Roles* they possess.

2.5.11.3. Dependency on State

A test with the specified *TestID* needs previously to have been successfully requested, and to have completed.

2.5.11.4. Effect on State

None.

2.5.11.5. Errors

Tab e 2-36: E C de f *GetTracerouteResult()*

e C de e	De c	De c
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
706	No Such Test	No test with the specified TestID was found.
707	Wrong Test Type	TestID is valid but refers to a different test type.
708	Invalid Test State	The TestID is valid but test results are not available.

2.5.12. *GetBandwidthTestInfo()*

The *GetBandwidthTestInfo()* action returns information about supported bandwidth tests, i.e. details of supported protocols, versions, profiles, settings, files and results. See section 2.3.27 for the definition of and examples of the returned information.

2.5.12.1. Arguments

Tab e 2-37: A g e f *GetBandwidthTestInfo()*

A g e	D ec	e a edS a eVa ab e
BandwidthTestInfo	*OUT*	*A_ARG_TYPE_Bandwidth-TestInfo*

2.5.12.2. Device Requirements

This action is *Non-Restrictable* and all control points MUST be permitted to invoke the action regardless of which *Roles* they possess.

The returned information MUST include all supported protocols, versions and profiles (as indicated by *Protocol* elements). This means that even if profile A includes all the requirements of profile B, both profiles A and B have to be reported independently. See section 2.6.6 for profile definitions.

The returned information can include passwords with which the bandwidth test client will authenticate itself with the bandwidth test server. When not running over TLS, empty strings SHOULD be returned in place of actual passwords.

2.5.12.3. Dependency on State

None.

2.5.12.4. Effect on State

None.

2.5.12.5. Errors

Tab e 2-38: E C de f *GetBandwidthTestInfo()*

e C de e De c	De c
400-499	See UPnP Device Architecture section on Control.
500-599	See UPnP Device Architecture section on Control.
600-699	See UPnP Device Architecture section on Control.

2.5.13. *BandwidthTest()*

The *BandwidthTest()* action requests a bandwidth test. If a bandwidth test is already active, the service MAY reject the request.

The input arguments are used as follows:

- : is an XML document that contains the test specification. See section 2.3.28 for the definition of and examples of the supplied information. The bandwidth test protocol, version and profile (as indicated by the *Protocol* element) MUST correspond to one of the supported bandwidth tests as reported via the *GetBandwidthTestInfo()* action (section 2.5.12).
- : indicates whether the recipient should play the role of server or of client.
- : is an XML document that specifies when the test is to be performed, whether it is to be repeated etc. This version of the specification does not define an XML Schema for the test schedule, so the value of this argument SHOULD be an empty string.
- : is the test's session ID. It is included in each protocol message and allows the recipient to determine the test with which the protocol message is associated. This version of the specification does not define any protocol-specific requirements for how this session ID should be used, so the value of this argument SHOULD be an empty string.

2.5.13.1. Arguments

Tab e 2-39: A g e f *BandwidthTest()*

A g e	D ec	e a edS a eVa ab e
BandwidthTestSpec	*IN*	*A_ARG_TYPE_Bandwidth-TestSpec*
TestEndpoint	*IN*	*A_ARG_TYPE_TestEnd-point*
TestSchedule	*IN*	*A_ARG_TYPE_TestSched-ule*
TestSessID	*IN*	*A_ARG_TYPE_TestSessID*

A g e	D ec	e a edS a eVa ab e
TestID	*OUT*	*A_ARG_TYPE_TestID*

2.5.13.2. Device Requirements

As specified by *DeviceProtection:1*, any control point that possesses any of the *Roles* in the action's *Role List* MUST be permitted to invoke this action regardless of the values of any action input arguments. If the *Security Feature* is not supported, all actions are permitted, i.e. behavior is the same as if the action had a *Role List* of "*Public*".

Otherwise, if all of the following conditions are met, any control point that possesses any of the *Roles* in the action's *Restricted Role List* MUST be permitted to invoke the action:

- The action was invoked over a TLS connection.
- The control point *Identity* is present in the *DeviceProtection:1* ACL.

2.5.13.3. Dependency on State

None.

2.5.13.4. Effect on State

When a bandwidth test is successfully requested, the *TestID* MUST be added to the *TestID* and *ActiveTestIDs* state variables.

2.5.13.5. Errors

Tab e 2-40: E C de f *BandwidthTest()*

e C de	e De c	De c
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.

- : is an XML document containing the result of the bandwidth test (this includes the test specification). See section 2.3.30 for the definition of and examples of the test result.

Unlike other diagnostic tests, bandwidth tests MAY allow partial results of an active or failed test to be returned. Bandwidth test profiles (section 2.6.6) can strengthen this "MAY" requirement to a "MUST" requirement.

2.5.14.1. Arguments

Tab e 2-41: A g e f *GetBandwidthTestResult()*

A g e	D ec	e a edS a eVa ab e
TestID	*IN*	*A_ARG_TYPE_TestID*
State	*OUT*	*A_ARG_TYPE_TestState*
Status	*OUT*	*A_ARG_TYPE_Bandwidth-TestStatus*
AdditionalInfo	*OUT*	*A_ARG_TYPE_String*
Result	*OUT*	*A_ARG_TYPE_Bandwidth-TestResult*

2.5.14.2. Device Requirements

This action is *Non-Restrictable* and all control points MUST be permitted to invoke the action regardless of which *Roles* they possess.

2.5.14.3. Dependency on State

A test with the specified *TestID* needs previously to have been successfully requested.

2.5.14.4. Effect on State

None.

2.5.14.5. Errors

Tab e 2-42: E C de f *GetBandwidthTestResult()*

e C de e De c	De c
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
706	No Such Test	No test with the specified TestID was found.
707	Wrong Test Type	TestID is valid but refers to a different test type.
708	Invalid Test State	The TestID is valid but test results are not available.

2.5.15. *InterfaceReset()*

The *InterfaceReset()* action requests that one or more IP interfaces should be reset. If an interface reset test is already active, the service MAY reject the request. If the test will reset the interface on which the request was received, the *Parent Device* MUST send the action response before initiating the test.

The input arguments are used as follows:

- : the IP interface or interfaces that are to be reset.

It is up to the implementation to decide what needs to be done in order to reset an IP interface. For example, if the interface's IP address was assigned via DHCP, it is almost certainly appropriate to release and renew the IP address. It might also be appropriate to reset the physical interface, clear out the DNS cache etc.

2.5.15.1. Arguments

Tab e 2-43: A g e f *InterfaceReset()*

A g e	D ec	e a edS a eVa ab e
Interfaces	*IN*	*A_ARG_TYPE_Interfaces*
TestID	*OUT*	*A_ARG_TYPE_TestID*

2.5.15.2. Device Requirements

As specified by *DeviceProtection:1*, any control point that possesses any of the *Roles* in the action's *Role List* MUST be permitted to invoke this action regardless of the values of any action input arguments. If the *Security Feature* is not supported, all actions are permitted, i.e. behavior is the same as if the action had a *Role List* of "*Public*".

Otherwise, if all of the following conditions are met, any control point that possesses any of the *Roles* in the action's *Restricted Role List* MUST be permitted to invoke the action:

- The action was invoked over a TLS connection.
- The control point *Identity* is present in the *DeviceProtection:1* ACL.
- The control point possesses a *Role* that authorizes use of the specified value of the *Interfaces* argument.

2.5.15.3. Dependency on State

None.

2.5.15.4. Effect on State

When an IP interface reset test is successfully requested, the *TestID* MUST be added to the *TestID* and *ActiveTestIDs* state variables.

Because an IP interface reset test can reset the interface on which the request was received, it might not be possible to read the test results until after a *Parent Device* restart. For this reason, IP interface reset test IDs MUST persist across such restarts.

Note that, regardless of the possibility of *Parent Device* restart, an event-driven control point will always discover that the test ID has been removed from *ActiveTestIDs*, either via an event generated when it is removed, or via the initial event when the control point subscribes after a *Parent Device* restart.

2.5.15.5. Errors

Tab e 2-44: E C de f *InterfaceReset()*

e C de	e De c	De c
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.

e C de	e De c	De c
606	Action not authorized	The action requested requires authorization and the sender was not authorized.
701	Interface Not Found	The requested IP interface was not found.
702	Interface Not Resettable	One or more of the requested IP interfaces has a static address and cannot be reset.
703	Test Already Active	A test of this type is already active (and the implementation doesn't support multiple active instances of a given test type).
704	Capabilities Precludes Test	Test arguments are individually valid but, taken together, describe a test that is beyond the service's capabilities.
705	State Precludes Test	Service state precludes performing this test.

2.5.16. GetInterfaceResetResult()

The *GetInterfaceResetResult()* action returns the results of a completed IP interface reset test. The output arguments are defined as follows:

- : indicates the overall success or failure of the test (if the test failed, the values of the remaining output arguments – other than – are not specified, and MUST be ignored).
- : a free-format string that can contain additional information about the test result.
- : The number of IP interfaces that were successfully reset.
- : The number of IP interfaces that could not be reset.

Note that, provided that the test did not fail, *NumberOfSuccesses* plus *NumberOfFailures* will always be the number of interfaces that *InterfaceReset()* requested to be reset.

2.5.16.1. Arguments

Tab e 2-45: A g e f GetInterfaceResetResult()

A g e	D ec	e a edS a eVa ab e
TestID	*IN*	*A_ARG_TYPE_TestID*
Status	*OUT*	*A_ARG_TYPE_Interface-ResetStatus*
AdditionalInfo	*OUT*	*A_ARG_TYPE_String*
NumberOfSuccesses	*OUT*	*A_ARG_TYPE_UShort*
NumberOfFailures	*OUT*	*A_ARG_TYPE_UShort*

2.5.16.2. Device Requirements

This action is *Non-Restrictable* and all control points MUST be permitted to invoke the action regardless of which *Roles* they possess.

2.5.16.3. Dependency on State

A test with the specified *TestID* needs previously to have been successfully requested, and to have completed.

2.5.16.4. Effect on State

None.

2.5.16.5. Errors

Table 2-46: Error Codes for GetInterfaceResetResult()

Error Code	Error Description	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
706	No Such Test	No test with the specified TestID was found.
707	Wrong Test Type	TestID is valid but refers to a different test type.
708	Invalid Test State	The TestID is valid but test results are not available.

2.5.17. SelfTest()

The *SelfTest()* action requests an implementation-specific self-test. If a self-test is already active, the service MAY reject the request.

2.5.17.1. Arguments

Table 2-47: Arguments for SelfTest()

Argument	Direction	relatedStateVariable
TestID	*OUT*	*A_ARG_TYPE_TestID*

2.5.17.2. Device Requirements

As specified by *DeviceProtection:1*, any control point that possesses any of the *Roles* in the action's *Role List* MUST be permitted to invoke this action regardless of the values of any action input arguments. If the *Security Feature* is not supported, all actions are permitted, i.e. behavior is the same as if the action had a *Role List* of "*Public*".

Otherwise, if all of the following conditions are met, any control point that possesses any of the *Roles* in the action's *Restricted Role List* MUST be permitted to invoke the action:

- The action was invoked over a TLS connection.
- The control point *Identity* is present in the *DeviceProtection:1* ACL.

2.5.17.3. Dependency on State

None.

2.5.17.4. Effect on State

When a self-test is successfully requested, the *TestID* MUST be added to the *TestID* and *ActiveTestIDs* state variables.

2.5.17.5. Errors

Table 2-48: Error Codes for SelfTest()

e C de	e De c	De c
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
606	Action not authorized	The action requested requires authorization and the sender was not authorized.
703	Test Already Active	A test of this type is already active (and the implementation doesn't support multiple active instances of a given test type).
705	State Precludes Test	Service state precludes performing this test.

2.5.18. GetSelfTestResult()

The GetSelfTestResult() action returns the results of a completed self-test. The output arguments are defined as follows:

- : indicates whether the test succeeded (_) or failed (_).
- : a free-format string that can contain additional information about the test result.

2.5.18.1. Arguments

Tab e 2-49: A g e f GetSelfTestResult()

A g e	D ec	e a edS a eVa ab e
TestID	IN	A_ARG_TYPE_TestID
Status	OUT	A_ARG_TYPE_Boolean
AdditionalInfo	OUT	A_ARG_TYPE_String

2.5.18.2. Device Requirements

This action is *Non-Restrictable* and all control points MUST be permitted to invoke the action regardless of which *Roles* they possess.

2.5.18.3. Dependency on State

A test with the specified TestID needs previously to have been successfully requested, and to have completed.

2.5.18.4. Effect on State

None.

2.5.18.5. Errors

Tab e 2-50: E C de f GetSelfTestResult()

e C de	e De c	De c
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.

e C de e De c	De c
706	No Such Test	No test with the specified TestID was found.
707	Wrong Test Type	TestID is valid but refers to a different test type.
708	Invalid Test State	The TestID is valid but test results are not available.

2.5.19. GetTestIDs()

The *GetTestIDs()* action returns a list of all known test IDs. A test ID is added to the list when a test is successfully requested, and is removed from the list when the implementation decides to delete all information associated with the test.

2.5.19.1. Arguments

Tab e 2-51: A g e f GetTestIDs()

A g e	D ec	e a edS a eVa ab e
TestIDs	*OUT*	*TestIDs*

2.5.19.2. Device Requirements

This action returns the value of an evented state variable. This value is freely available to all control points, so the action is *Non-Restrictable* and all control points MUST be permitted to invoke the action regardless of which *Roles* they possess.

2.5.19.3. Dependency on State

None.

2.5.19.4. Effect on State

None.

2.5.19.5. Errors

Tab e 2-52: E C de f GetTestIDs()

e C de	e De c	De c
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.

2.5.20. GetActiveTestIDs()

The *GetActiveTestIDs()* action returns a list of the test IDs associated with active tests. A test ID is added to the list when a test is successfully requested, and is removed from the list when the test completes, whether successfully or unsuccessfully, or is canceled.

2.5.20.1. Arguments

Tab e 2-53: A g e f GetActiveTestIDs()

A g e	D ec	e a edS a eVa ab e
TestIDs	*OUT*	*ActiveTestIDs*

2.5.20.2. Device Requirements

This action returns the value of an evented state variable. This value is freely available to all control points, so the action is *Non-Restrictable* and all control points MUST be permitted to invoke the action regardless of which *Roles* they possess.

2.5.20.3. Dependency on State

None.

2.5.20.4. Effect on State

None.

2.5.20.5. Errors

Tab e 2-54: E C de f GetActiveTestIDs()

e C de	e De c	De c
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.

2.5.21. GetTestInfo()

The *GetTestInfo()* action returns the type and state of a successfully requested test.

2.5.21.1. Arguments

Tab e 2-55: A g e f GetTestInfo()

A g e	D ec	e a edS a eVa ab e
TestID	*IN*	*A_ARG_TYPE_TestID*
Type	*OUT*	*A_ARG_TYPE_TestType*
State	*OUT*	*A_ARG_TYPE_TestState*

2.5.21.2. Device Requirements

This action is *Non-Restrictable* and all control points MUST be permitted to invoke the action regardless of which *Roles* they possess.

2.5.21.3. Dependency on State

A test with the specified *TestID* needs previously to have been successfully requested.

2.5.21.4. Effect on State

None.

2.5.21.5. Errors

Tab e 2-56: E C de f GetTestInfo()

e C de	e De c	De c
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.

e C de	e De c	De c
600-699	TBD	See UPnP Device Architecture section on Control.
706	No Such Test	No test with the specified TestID was found.

2.5.22. *CancelTest()*

The *CancelState()* action cancels a successfully requested test.

2.5.22.1. Arguments

Tab e 2-57: A g e f *CancelTest()*

A g e	D ec	e a edS a eVa ab e
TestID	*IN*	*A_ARG_TYPE_TestID*

2.5.22.2. Device Requirements

As specified by *DeviceProtection:1*, any control point that possesses any of the *Roles* in the action's *Role List* MUST be permitted to invoke this action regardless of the values of any action input arguments. If the *Security Feature* is not supported, all actions are permitted, i.e. behavior is the same as if the action had a *Role List* of "*Public*".

Otherwise, if all of the following conditions are met, any control point that possesses any of the *Roles* in the action's *Restricted Role List* MUST be permitted to invoke the action:

- The action was invoked over a TLS connection.
- The control point *Identity* is present in the *DeviceProtection:1* ACL.
- The control point initiated the test.

2.5.22.3. Dependency on State

A test with the specified *TestID* needs previously to have been successfully requested, and not to have completed.

2.5.22.4. Effect on State

When a test is successfully canceled, the *TestID* MUST be removed from the *ActiveTestIDs* state variable.

2.5.22.5. Errors

Tab e 2-58: E C de f *CancelTest()*

e C de	e De c	De c
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
606	Action not authorized	The action requested requires authorization and the sender was not authorized.
706	No Such Test	No test with the specified TestID was found.
709	State Precludes Cancel	The TestID is valid but the test can't be canceled.

2.5.23. GetLogURIs()

The *GetLogURIs()* action retrieves a list of URIs for the logs currently supported by the *Parent Device*. Logs can potentially be added to or removed from this list at run-time, although the mechanism via which this might happen is implementation-specific.

2.5.23.1. Arguments

Tab e 2-59: A g e f GetLogURIs()

A g e	D ec	e a edS a eVa ab e
LogURIs	*OUT*	*LogURIs*

2.5.23.2. Device Requirements

This action returns the value of an evented state variable. This value is freely available to all control points, so the action is *Non-Restrictable* and all control points MUST be permitted to invoke the action regardless of which *Roles* they possess.

2.5.23.3. Dependency on State

None.

2.5.23.4. Effect on State

None.

2.5.23.5. Errors

Tab e 2-60: E C de f GetLogURIs()

e C de	e De c	De c
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.

2.5.24. SetLogInfo()

The *SetLogInfo()* action enables / disables the specified log and sets its log level.

2.5.24.1. Arguments

Tab e 2-61: A g e f SetLogInfo()

A g e	D ec	e a edS a eVa ab e
LogURI	*IN*	*A_ARG_TYPE_LogURI*
Enabled	*IN*	*A_ARG_TYPE_Boolean*
LogLevel	*IN*	*A_ARG_TYPE_LogLevel*

2.5.24.2. Device Requirements

As specified by *DeviceProtection:1*, any control point that possesses any of the *Roles* in the action's *Role List* MUST be permitted to invoke this action regardless of the values of any action input arguments. If the *Security Feature* is not supported, all actions are permitted, i.e. behavior is the same as if the action had a *Role List* of "*Public*".

Otherwise, if all of the following conditions are met, any control point that possesses any of the *Roles* in the action's *Restricted Role List* MUST be permitted to invoke the action:

- The action was invoked over a TLS connection.
- The control point *Identity* is present in the *DeviceProtection:1* ACL.
- The control point possesses a *Role* that authorizes use of the specified value of the *LogURI* argument.

2.5.24.3. Dependency on State

LogURI needs to identify one of the logs currently supported by the *Parent Device*.

2.5.24.4. Effect on State

The enable/disable and level settings associated with the log identified by *LogURI* are changed. These values MUST persist across *Parent Device* restarts. Entries will no longer be written to a disabled log. It is up to the implementation to decide whether disabling a log will clear out any existing entries.

2.5.24.5. Errors

Table 2-62: Error Codes for *SetLogInfo()*

Error Code	Error Description	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
606	Action not authorized	The action requested requires authorization and the sender was not authorized.
710	No Such Log	No log with the specified LogURI was found.
711	Log Not Configurable	Log doesn't permit enable/disable and/or log level to be changed.

2.5.25. *GetLogInfo()*

The *GetLogInfo()* action returns information about the specified log.

- *Configurable* indicates whether the log is configurable. A log is configurable if it can be enabled/-disabled and/or if its log level can be changed.
- A *MaxSize* of 0 indicates that the maximum possible size is not fixed or is not known (section 2.3.36).

2.5.25.1. Arguments

Table 2-63: Arguments for *GetLogInfo()*

Argument	Direction	relatedStateVariable
LogURI	*IN*	*A_ARG_TYPE_LogURI*
Configurable	*OUT*	*A_ARG_TYPE_Boolean*
Enabled	*OUT*	*A_ARG_TYPE_Boolean*
LogLevel	*OUT*	*A_ARG_TYPE_LogLevel*
LogURL	*OUT*	*A_ARG_TYPE_LogURL*

A g e	D ec	e a edS a eVa ab e
MaxSize	*OUT*	*A_ARG_TYPE_LogMaxSize*
LastChange	*OUT*	*A_ARG_TYPE_DateTime*

2.5.25.2. Device Requirements

This action is *Non-Restrictable* and all control points MUST be permitted to invoke the action regardless of which *Roles* they possess.

2.5.25.3. Dependency on State

LogURI needs to identify one of the logs currently supported by the *Parent Device*.

2.5.25.4. Effect on State

None.

2.5.25.5. Errors

Tab e 2-64: E C de f *GetLogInfo()*

e C de	e De c	De c
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
710	No Such Log	No log with the specified LogURI was found.

2.5.26. *GetACLData()*

The *GetACLData()* action returns the *BasicManagement:2* access control list for the *Parent Device*. This access control list gives the control point information about how the device will make access control decisions. The syntax of the access control list is described in section 2.3.37.

2.5.26.1. Arguments

Tab e 2-65: A g e f *GetACLData()*

A g e	D ec	e a edS a eVa ab e
ACL	*OUT*	*A_ARG_TYPE_ACL*

2.5.26.2. Device Requirements

As specified by *DeviceProtection:1*, any control point that possesses any of the *Roles* in the action's *Role List* MUST be permitted to invoke this action regardless of the values of any action input arguments. If the *Security Feature* is not supported, all actions are permitted, i.e. behavior is the same as if the action had a *Role List* of "*Public*".

Otherwise, if all of the following conditions are met, any control point that possesses any of the *Roles* in the action's *Restricted Role List* MUST be permitted to invoke the action:

- The action was invoked over a TLS connection.
- The control point *Identity* is present in the *DeviceProtection:1* ACL.

2.5.26.3. Effect on State

None.

2.5.26.4. Errors

Tab e 2-66: E C de f *GetACLData()*

e C de	e De c	De c
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
606	Action not authorized	The action requested requires authorization and the sender was not authorized.

2.5.27. C E C de

The following table lists error codes common to actions for this service type. If an action results in multiple errors, the most specific error MUST be returned.

Tab e 2-67: C E C de

e C de	e De c	De c
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
606	Action not authorized	The action requested requires authorization and the sender was not authorized.
700		Reserved for future extensions.
701	Interface Not Found	The requested IP interface was not found.
702	Interface Not Resettable	One or more of the requested IP interfaces has a static address and cannot be reset (InterfaceReset).
703	Test Already Active	A test of this type is already active (and the implementation doesn't support multiple active instances of a given test type).
704	Capabilities Preclude Test	Test arguments are individually valid but, taken together, describe a test that is beyond the service's capabilities.
705	State Precludes Test	Service state precludes performing this test.
706	No Such Test	No test with the specified TestID was found.
707	Wrong Test Type	TestID is valid but refers to a different test type.
708	Invalid Test State	The TestID is valid but test results are not available.
709	State Precludes Cancel	The TestID is valid but the test can't be canceled.
710	No Such Log	No log with the specified LogURI was found.

e C de	e De c	De c
711	Log Not Configurable	Log doesn't permit enable/disable and/or log level to be changed.
712	Invalid Test Endpoint	The service does not support this test for the specified endpoint.

800-

2.6. Ba d d h Te (N a e)

Bandwidth tests are mentioned in several sections of this document. Firstly, section 2.2.2.1 gives a high-level overview; it explains the three bandwidth test actions, each of which can be invoked on either the server or the client, and each of which either supplies or returns an appropriate XML document:

- *GetBandwidthTestInfo()* returns a *BandwidthTestInfo* XML document.
- *BandwidthTest()* supplies a *BandwidthTestSpec* XML document.
- *GetBandwidthTestResult()* returns a *BandwidthTestResult* XML document.

Sections 2.3.27, 2.3.28 and 2.3.30 define and illustrate the structure of these three XML documents, and sections 2.5.12, 2.5.13 and 2.5.14 define the three bandwidth test actions. The XML Schema (section 4) is the normative definition of the XML documents but does not (and can not) define all the detailed usage rules.

This section provides the formal definitions of the various parameters (whether capabilities, settings or results) that can occur in the XML documents. It also defines some profiles, each of which corresponds to a bandwidth test that meets a specific set of requirements.

This section contains very little additional information about the actions, because they behave pretty much the same as do the other diagnostic test actions.

Finally, note that section 2.7 provides several examples of bandwidth tests.

2.6.1. A ach

The information in the bandwidth test XML documents corresponds to the following [EBNF]. This EBNF is non-normative but is believed to correspond exactly to the normative XML Schema. Please note the following:

- The EBNF does not contain definitions for individual capabilities (section 2.6.3), settings (section 2.6.4) and results (section 2.6.5). These terms are shown in *italics*.
- There are quite a lot of possibilities, so for ease of reading the most commonly used terms are shown in .

BandwidthTestInfo	**::=**	**(ProtocolKey + Endpoint + EndpointInfo)+**
BandwidthTestSpec	**::=**	**ProtocolKey + SpecSettings**
BandwidthTestResult	**::=**	**ProtocolKey + SpecSettings + Endpoint + Results**
ProtocolKey	**::=**	**Protocol** *Version* **Profile**
Protocol	**::=**	**/* protocol name, e.g. "HTTP" or "Iperf" */**
Endpoint	**::=**	**/* Server or Client */**
EndpointInfo	**::=**	**SettingsCaps + FilesCaps + ResultsCaps**
SettingsCaps	**::=**	**SettingCap(Settings)**

FilesCaps	**::=**	**File***
File	**::=**	***Path* + *RealFile*? + *SizeBytes*? + MIME*Type*?**
ResultsCaps	**::=**	**ResultCap(Results)**
SpecSettings	**::=**	**Settings**
Settings	**::=**	***Interface? + Transport*? + *Host*? + *Port*? + *TLSPort*? + *SockBuffBytes*? + *AppBuffBytes*? + *Direction*? + *TimeMSecs? + LengthBytes? + Username? + Password? + Path? + BandwidthBytesPerSec*? + *DSCP*? + *EthernetPriority*? + *IntervalMSecs*? + *Threads*? + *TCPNoDelay*? + *TCPMSS*? + *ExitWhenDone*?**
Results	**::=**	***TCPSYNTime*? + *TCPSYNACKTime*? + *TCPLateACKs*? + *ROMTime*? + *BOMTime*? + *EOMTime*? + *TestBytesSent*? + *TestBytesReceived*? + *TotalBytesSent*? + *TotalBytesReceived*? + *TestPacketsSent*? + *TestPacketsReceived*? + *TotalPacketsSent*? + *TotalPacketsReceived*?**

		+ *DuplicatePackets*? + *LostPackets*? + *OutOfSequencePackets*? + *ResentPackets*? + *LatencyUSecs*? + *JitterUSecs*?

The EBNF probably seems rather forbidding, so *BandwidthTestInfo*, *BandwidthTestSpec* and *BandwidthTestResult* are illustrated separately. The EBNF below is the same as the above full EBNF but less common terms have been removed and a few definitions have been collapsed.

BandwidthTestInfo

BandwidthTestInfo	**::=**	**(Protocol + Endpoint + SettingsCaps + FilesCaps + ResultsCaps)+**
Protocol	**::=**	**/* protocol name, e.g. "HTTP" or "Iperf" */**
Endpoint	**::=**	**/* Server or Client */**
SettingsCaps	**::=**	**SettingCap(Settings)**
FilesCaps	**::=**	**File***
File	**::=**	**Path + RealFile? + SizeBytes? + MIMEType?**
ResultsCaps	**::=**	**ResultCap(Results)**
Settings	**::=**	***Interface? + Transport*? + *Host*? + *Port*? + *TLSPort*? + *SockBuffBytes*? + *AppBuffBytes*? + *Direction*? + *TimeMSecs? + LengthBytes? + Username*? + *Password*? + *Path*? + *BandwidthBytesPerSec*? + *DSCP*? + *EthernetPriority*? + *IntervalMSecs*? + *Threads*? + *TCPNoDelay*? + *TCPMSS*? + *ExitWhenDone*?**
Results	**::=**	***TCPSYNTime*? + *TCPSYNACKTime*? + *TCPLateACKs*? + *ROMTime*? + *BOMTime*? + *EOMTime*? + *TestBytesSent*? + *TestBytesReceived*? + *TotalBytesSent*? + *TotalBytesReceived*? + *TestPacketsSent*? + *TestPacketsReceived*? + *TotalPacketsSent*? + *TotalPacketsReceived*? + *DuplicatePackets*? + *LostPackets*? + *OutOfSequencePackets*? + *ResentPackets*? + *LatencyUSecs*? + *JitterUSecs*?**

BandwidthTestSpec

BandwidthTestSpec	**::=**	**Protocol + Settings**

BandwidthTestResult

BandwidthTestResult	**::=**	**Protocol + Settings + Endpoint + Results**

The XML Schema is structured very similarly to the EBNF and uses the same internal names, but of course it uses XML-specific concepts. For example, this is a valid *BandwidthTestResult* that clearly matches the corresponding EBNF.

BandwidthTestResult

Protocol

Settings

Endpoint

Results

2.6.2. P c

ProtocolKey	**::=**	**Protocol** *Version* **Profile**

Bandwidth test protocols are identified by name and have an associated version and profile. An implementation can support multiple versions and profiles for a given protocol.

Tab e 2-68: Ba d d h Te P c

Pa a e e	T e	De c	Defa
Protocol		Required protocol name. The following protocol names are defined in this specification. • *HTTP*: RFC 2616 [HTTP] • *FTP*: RFC 959 [FTP] • *Iperf*: [IPERF] • *Echo*: RFC 862 [ECHO] • *EchoPlus*: [TR-143] Iperf v3 is incompatible with Iperf v2 and is (at the time of writing) less actively developed than Iperf v2. Therefore Iperf v2 is preferred.	-
Version		Optional protocol version. Defines a supported protocol version. A physical device or control point can support more than one version of a given protocol. Syntax is not constrained by the XML Schema but if a version is specified, it SHOULD be of the form *n.m*…, i.e. match the regular expression *\d+(\.\d+)**	“”

Pa a e e	T e	De c	Defa
Profile		Required protocol profile. Corresponds to a named set of additional protocol-specific requirements. For each protocol, a mandatory *Baseline* profile defines requirements that have to be met by all device implementations. See section 2.6.6.	-

2.6.3. Ca ab e

```
EndpointInfo ::= SettingsCaps + FilesCaps + ResultsCaps
SettingsCaps ::= SettingCap(Settings)
FilesCaps    ::= File*
File         ::= Path + RealFile? + SizeBytes? + MIMEType?
ResultsCaps  ::= ResultCap(Results)
```

Capabilities indicate the following:

- Which settings are supported, whether they are settable, and the supported values or range.
- Which files are supported, and their attributes.
- Which results are supported.

If a particular capability is not indicated, its default value (if specified) is assumed.

Pa a e e	T e	De c	Defa
Settings (*Interface*, *Transport*, *Port*, *TLSPort* etc)	*Various*	Settings capabilities indicate which settings are supported, whether they are settable, and the supported values or range. Individual settings are defined in section 2.6.4.	*Various*
File[*]/*Path*		Required relative path that identifies a data source or destination. The path is formatted using "/" delimiters as specified in [URI]. It is relative to the directory that the server or client has chosen to use for bandwidth test data. For example the server might serve content from the directory */upnp/data*, in which case the path *test.dat* would refer to the file */upnp/data/test.dat*. If Path ends with a "/" character, it indicates a directory sub-tree.	-
File[*]/*RealFile*		Optional indication of whether the data source or destination identified by *Path* is a real file (as opposed to being a source of auto-generated data or a sink for discarded data). If Path ends with a "/" character, *RealFile* applies to the directory sub-tree.	_
File[*]/*SizeBytes*		Optional file size in bytes. Only applies to real files.	-

Pa a e e	T e	De c	Defa
File[*]/*MIMEType*		Optional file MIME type.	-
Results (*TCPSYN-Time*, *TCPSYNACK-Time*, *TCPLateACKs* etc)	*Various*	Results capabilities only indicate which results are supported. Individual results are defined in section 2.6.x.	*n/a*

2.6.4. Se g

```
Settings  ::=  Interface? + Transport? + Host? + Port? + TLSPort? +
                  SockBuffBytes? + AppBuffBytes? + Direction? +
                  TimeMSecs? + LengthBytes? + Username? + Password? +
                  Path? + BandwidthBytesPerSec? + DSCP? +
                  EthernetPriority? + IntervalMSecs? + Threads? +
                  TCPNoDelay? + TCPMSS? + ExitWhenDone?
```

Settings can be used in three contexts:

- As a capability in *BandwidthTestInfo*.
- As a setting in *BandwidthTestSpec*.
- As a setting in *BandwidthTestResult* (because the result includes the spec).

If a particular setting is not provided, its default value (if specified) is assumed.

All settings are optional but profiles can add requirements.

Pa a e e	T e	De c	Defa
Interface		IP interface name. This is the name by which the interface is known in the [CMS] data model, i.e. the value of the corresponding */UPnP/DM/Configuration/Network/IPInterface/#/SystemName* parameter.	-
Transport		Layer 4 (transport) protocol. This specification only considers *TCP* and *UDP*, but additional transport protocols MAY be specified using their standard acronyms.	-
Host		Host name or IP address. For a server, this determines the IP address to which the listening socket will be bound. For a client, this determines the server to which the client will connect.	-
Port *TLSPort*		Port number. For both client and server, this is a port on which the program will listen and accept connections or traffic.	-
SockBuffBytes		Socket buffer size in bytes. Set using the *setsockopt(SO_SNDBUF/SO_RCVBUF)* system call. For *TCP* this is the TCP window.	-
AppBuffBytes		Application buffer size in bytes. This is the length of the buffer that is used for *read()/write()* system calls.	-

Pa a e e	T e	De c	Defa
Direction		The direction, from the client point of view, in which test data will travel. Can be: • *Put*: data will be sent from client to server. • *Get*: data will be sent from server to client. • *PutEcho*: data will be sent from client to server, then echoed back to client.	-
TimeMSecs *LengthBytes*		Determines the length of the test: • *TimeMSecs*: the length of time in milliseconds. • *LengthBytes*: the number of bytes of test data that will be transferred. Not more than one of *TimeMSecs* and *LengthBytes* can be specified. If neither is specified, *Path* MUST be specified, and implicitly determines *LengthBytes*. If *Path* is specified and the end of the referenced file is encountered before *TimeMSecs* / *LengthBytes* has been satisfied, it is up to the protocol and/or implementation to determine whether and how the test will continue. Possibilities include: • The test ends in error. • The test ends early (it is as though *LengthBytes* had been set to the file length). • The test continues, and the last application buffer is re-used as many times as necessary. • The test continues, re-reading the file from the beginning.	- -
Username *Password*		The credentials with which a client authenticates itself to a server. This might be extended, e.g. to support certificates, in a future version.	- -
Path		A relative path that identifies a data source or sink. See the definition of the *Path* capability in section 2.6.3.	-
BandwidthBytes-PerSec		The desired bandwidth in bytes per second. If this is not specified, data will be sent as fast as possible.	-
DSCP		The Diffserv code point for marking packets transmitted in the test.	0
EthernetPriority		The Ethernet priority code for marking packets transmitted in the test.	0
IntervalMSecs		The interval in milliseconds over which results are collected. For example, if *TimeMSecs* is 10000 and *IntervalMSecs* is 2000, the results for intervals [0:2000], [2000:4000], [4000:6000], [6000:8000]	-

Pa a e e	T e	De c	Defa
		and [8000:10000] will be collected.	
Threads		The number of data threads on the client or server. For the client, this is number of parallel invocations of the test. For the server, this is the number of threads that can receive data.	-
TCPNoDelay		Applies only to TCP. Controls whether to disable Nagle's algorithm, which is done using *setsockopt(TCP_NODELAY)*.	_
TCPMSS		Applies only to TCP. Controls the maximum segment size, which is done using *setsockopt(TCP_MAXSEG)*.	-
ExitWhenDone		A non-binding request to a server or client to exit once a test has completed.	_

2.6.5. Re

```
Results  ::=  TCPSYNTime? + TCPSYNACKTime? + TCPLateACKs? + ROMTime? +
              BOMTime? + EOMTime? + TestBytesSent? +
              TestBytesReceived? + TotalBytesSent? +
              TotalBytesReceived? + TestPacketsSent? +
              TestPacketsReceived? + TotalPacketsSent? +
              TotalPacketsReceived? + DuplicatePackets? +
              LostPackets? + OutOfSequencePackets? + ResentPackets?
              + LatencyUSecs? + JitterUSecs?
```

All results are optional but profiles can add requirements.

All values SHOULD be specified to microsecond precision, e.g. *2008-04-09T15:01:05.-123456*.

Pa a e e	T e	De c
TCPSYNTime		Applies only to *TCP*. The absolute time at which the client sent (or the server received) the initial SYN for the *TCP* connection over which the test data will flow.
TCPSYNACKTime		Applies only to *TCP*. The absolute time at which the client received (or the server sent) the ACK to the initial SYN for the *TCP* connection over which the test data will flow.
TCPLateACKs		Applies only to *TCP*. The number of occasions on which an ACK was judged to be late (by the client or the server) for the *TCP* connection over which the test data is flowing. It is up to the implementation to decide how to interpret "late". Note that http://www.stuartcheshire.org/papers/Nagle-DelayedAck provides useful background discussion and suggests that a late ACK might be defined as one that takes over 100ms.
ROMTime		The absolute time at which the client sent (or the server received) the first protocol-specific request on the

Pa a e e	T e	De c
		connection over which the test data will flow.
BOMTime		The absolute time at which the client (or server) first knew that test data transmission could begin. This is worded carefully in order to cover protocols where there is no initial negotiation (e.g. *HTTP*, *Echo*) and protocols where is there is an initial negotiation (e.g. *FTP*).
EOMTime		The absolute time at which the client (or server) first knew that all the test data had been sent. This is worded carefully in order to cover protocols where there is no explicit indication of end of transfer (e.g. *HTTP*, *Echo*) and protocols where is there is such an explicit indication (e.g. *FTP*).
TestBytesSent *TestBytesReceived*		The number bytes of test data that were transferred over the data connection.
TotalBytesSent *TotalBytesReceived*		The total number of IP payload bytes that were transferred over the data connection.
TestPacketsSent *TestPacketsReceived*		The number of protocol-specific packets that were transferred over the data connection.
TotalPacketsSent *TotalPacketsReceived*		The number of IP packets that were transferred over the data connection.
DuplicatePackets		Applies only to unreliable transports such as *UDP*. Also applies only to the recipient of the test data. The number of packets received over the data connection that were duplicates of previously received packets. Note that RFC 4689 [TERMS] defines a Duplicate Packet as (assuming that each packet has an incrementing sequence number) "*A received packet with a Test Sequence number matching a previously received packet*".
LostPackets		Applies only to unreliable transports such as *UDP*. Also applies only to the recipient of the test data. The number of packets that were never received over the data connection. It is up to the implementation to decide how to interpret "never received".
OutOfSequencePackets		Applies only to unreliable transports such as *UDP*. Also applies only to the recipient of the test data. The number of packets received over the data connection that were out of sequence. It is up to the implementation to decide how to interpret "out of sequence". Note that RFC 4689 [TERMS] defines an Out-Of-Order Packet as (assuming that each packet has an incrementing sequence number) "*A received packet with a sequence number less than the sequence number of any previously arriving packet*".
ResentPackets		Applies only to unreliable transports such as *UDP*. Also applies only to the sender of the test data. The number of packets sent over the data connection that subsequently had to be re-sent. It is up to the implementation to decide how to interpret "re-sent" but it is RECOMMENDED that if the

Pa a e e	T e	De c
		same packet is re-sent *n* times then the counter is incremented by *n*.
LatencyUSecs		The latency in microseconds when delivering test data packets. All other details are determined by the protocol and/or the implementation. Note that latency can be one-way or round-trip (which is of course much easier to measure). It will be common for the measurement to be the maximum round-trip latency.
JitterUSecs		The jitter in microseconds when delivering test data packets. All other details are determined by the protocol and/or the implementation. Note that jitter is usually regarded as an unsigned quantity, e.g. RFC 4689 [TERMS] defines it as "*The absolute value of the difference between the Forwarding Delay of two consecutive received packets belonging to the same stream*". However, for maximum flexibility, the data type allows it to be negative.

2.6.6. P f e

The XML Schema does not specify which settings and results are mandatory for a given protocol. This is deliberate: the settings and results are a "toolkit" that is designed to apply to a wide range of bandwidth test protocols, but a given protocol might require only a small number of "tools".

Profiles are introduced as a way of associating specific requirements with a given protocol. If a device reports that it implements a given profile, the control point can assume that it supports the requirements associated with that profile. Profiles therefore improve interoperability.

The following profiles are defined in this specification.

- *Baseline*: mandatory profile that is defined for the *HTTP*, *FTP*, *Echo*, *EchoPlus* and *Iperf* protocols, i.e. for all the protocols that are defined in this document; specifies a set of basic requirements.
- *BBF*: extended profile that is defined for the *HTTP*, *FTP*, *Echo* and *EchoPlus* protocols; specifies additional Broadband Forum [TR-143] requirements.

As explained in section 2.6.7, vendors, other working committees, other organizations, and future UPnP DM versions can define additional profiles.

The requirements tables all have an column, which indicates whether the parameter is a setting ( ) or a result ( ).

For all settings ( ) listed in the table:

- The setting MUST be included in *BandwidthTestInfo*.
- The setting's allowed values and/or ranges MUST (*unless inappropriate*) be included in *BandwidthTestInfo*.
- The setting MUST (*if applicable*) be settable via *BandwidthTestSpec*.
- The setting MUST (*if applicable*) be included in *BandwidthTestResult*.

For all results ( ) listed in the table:

- The result MUST be included in *BandwidthTestInfo*.
- The result MUST (*if applicable*) be included in *BandwidthTestResult*.

The "*unless inappropriate*" qualification permits allowed values and/or ranges to be omitted for settings such as *Host*, for which the implementation is unlikely to know the full set of possibilities, or *Password*, for which the implementation is (very) unlikely to wish to report the full set of possibilities.

The "*if applicable*" qualification allows omission of parameters that don't apply to a given test. For example:

- If *TimeMSecs* is specified, *LengthBytes* has to be omitted because only one of *TimeMSecs* and *LengthBytes* can be specified. This is stated in the settings definitions in section 2.6.4.
- *Iperf JitterUSecs* can be omitted if the *Transport* is *TCP* because it applies only to *UDP* tests. This is stated in the *Iperf* profile definition in section 2.6.6.3.

2.6.6.1. HTTP and FTP Profiles

These basic requirements apply when *Protocol* is *HTTP* or *FTP* and *Profile* is *Baseline*.

Pa a e e	Se g/ Re	C e
Endpoint		MUST support *Server* and/or *Client*
Transport	S	MUST support *TCP*
Host	S	A value MUST be supplied in *BandwidthTestSpec* This is always the host to which the client will connect
Port	S	This is always the port to which the client will connect
Direction	S	MUST support *Get* (download) and/or *Put* (upload)
LengthBytes	S	MUST be supported if *Direction Put* (upload) is supported
Path	S	A value MUST be supplied in *BandwidthTestSpec*
TotalBytesSent	R	MUST be supported by *Client* if *Direction Put* (upload) is supported MUST be supported by *Server* if *Direction Get* (download) is supported
TotalBytesReceived	R	MUST be supported by *Client* if *Direction Get* (download) is supported MUST be supported by *Server* if *Direction Put* (upload) is supported

These requirements (a superset of the *Baseline* requirements) apply when *Protocol* is *HTTP* or *FTP* and *Profile* is *BBF*.

Pa a e e	Se g/ Re	C e
Endpoint		MUST support *Client*

Pa a e e	Se g/ Re	C e
Protocol		If *HTTP* is supported, persistent connections MUST be used, pipelining MUST NOT be used, and HTTP authentication MUST NOT be used If *FTP* is supported, binary transfers MUST be used
Transport	S	MUST support *TCP*
Host	S	A value MUST be supplied in *BandwidthTestSpec* This is always the host to which the client will connect
Port	S	This is always the port to which the client will connect

Direction

Pa a e e	Se g/ Re	C e
TestBytesReceived	R	MUST be supported by *Client* if *Direction* *Get* (download) is supported MUST be supported by *Server* if *Direction* *Put* (upload) is supported The test traffic received in bytes during the FTP/HTTP transaction including FTP/HTTP headers, between *BOMTime* and *EOMTime*
TotalBytesSent	R	MUST be supported by *Client* if *Direction* *Put* (upload) is supported MUST be supported by *Server* if *Direction* *Get* (download) is supported The total number of bytes sent on the interface between *BOMTime* and *EOMTime*
TotalBytesReceived	R	MUST be supported by *Client* if *Direction* *Get* (download) is supported MUST be supported by *Server* if *Direction* *Put* (upload) is supported The total number of bytes received on the interface between *BOMTime* and *EOMTime*

2.6.6.2. Echo and EchoPlus Profiles

These basic requirements apply when *Protocol* is *Echo* or *EchoPlus* and *Profile* is *Baseline*.

Pa a e e	Se g/ Re	C e
Endpoint		MUST support *Server*
Protocol		*GetBandwidthTestResult()* MUST support the return of partial results while the test is still active.
Transport	S	MUST support *UDP*
Port	S	
Direction	S	MUST support *PutEcho*

Pa a e e	Se g/ Re	C e
Transport	S	MUST support *UDP*
Host	S	A value MUST be supplied in *BandwidthTestSpec* This is the Source IP address of the echo packet The device MUST only respond to a echo from this source IP address
Port	S	The port on which the server MUST listen and respond to echo requests
Direction	S	MUST support *PutEcho* This is implied by the protocol, so it MAY be omitted from *BandwidthTestInfo*, *BandwidthTestSpec* and *BandwidthTestResult*
BOMTime	R	(TR-143 parameter is TimeFirstPacketReceived) The time that the server received the first echo packet
EOMTime	R	(TR-143 parameter is TimeLastPacketReceived) The time that the server received the most recent echo packet
TotalBytesReceived *TotalBytesSent*	R	(TR-143 parameters are BytesReceived and BytesResponded) BytesReceived: the number of received bytes including payload and header BytesResponded: the number of responded bytes, including payload and header
TotalPacketsReceived *TotalPacketsSent*	R	(TR-143 parameters are PacketsReceived and PacketsResponded) PacketsReceived: incremented upon each valid echo packet received PacketsResponded: incremented for each echo response sent

2.6.6.3. Iperf profiles

These basic requirements apply when *Protocol* is *Iperf* and *Profile* is *Baseline*.

Pa a e e	Se g/ Re	C e
Endpoint		MUST support *Server* and/or *Client*
Transport	S	MUST support *TCP* and *UDP*
Host	S	A value MUST be supplied in *BandwidthTestSpec* If within a <*Server*> XML element, this determines the IP address to which the server will bind; otherwise it is the host to which the client will connect
Port	S	If within a <*Client*> XML element, this is the port on which the client will listen; otherwise it is the port on which the server will listen and to which the client will connect
SockBuffBytes	S	
AppBuffBytes	S	
Direction	S	Only used by client; MUST support *Put* and *PutEcho*

Pa a e e	Se g/ Re	C e
TimeMSecs	S	Only used by client
LengthBytes	S	Only used by client
BandwidthBytesPerSec	S	Only used by client; only applies to *UDP* tests
IntervalMSecs	S	
Threads	S	Defaults to Iperf default
TCPNoDelay	S	
TCPMSS	S	
ExitWhenDone	S	
TestBytesSent *TestBytesReceived*	R	
TestPacketsSent *TestPacketsReceived*	R	Only applies to *UDP* tests
LostPackets	R	Only applies to *UDP* tests
JitterUSecs	R	Only applies to *UDP* *PutEcho* tests

2.6.7. E e b

Vendors, other working committees, other organizations, and future UPnP DM versions can define additional bandwidth test protocols, profiles, settings and results using the usual UPnP XML extension rules. Naming conventions for XML elements and attributes are as specified by [UDA1.0], but naming conventions for additional protocols (and other enumerated values, e.g. transport protocols) are laxer:

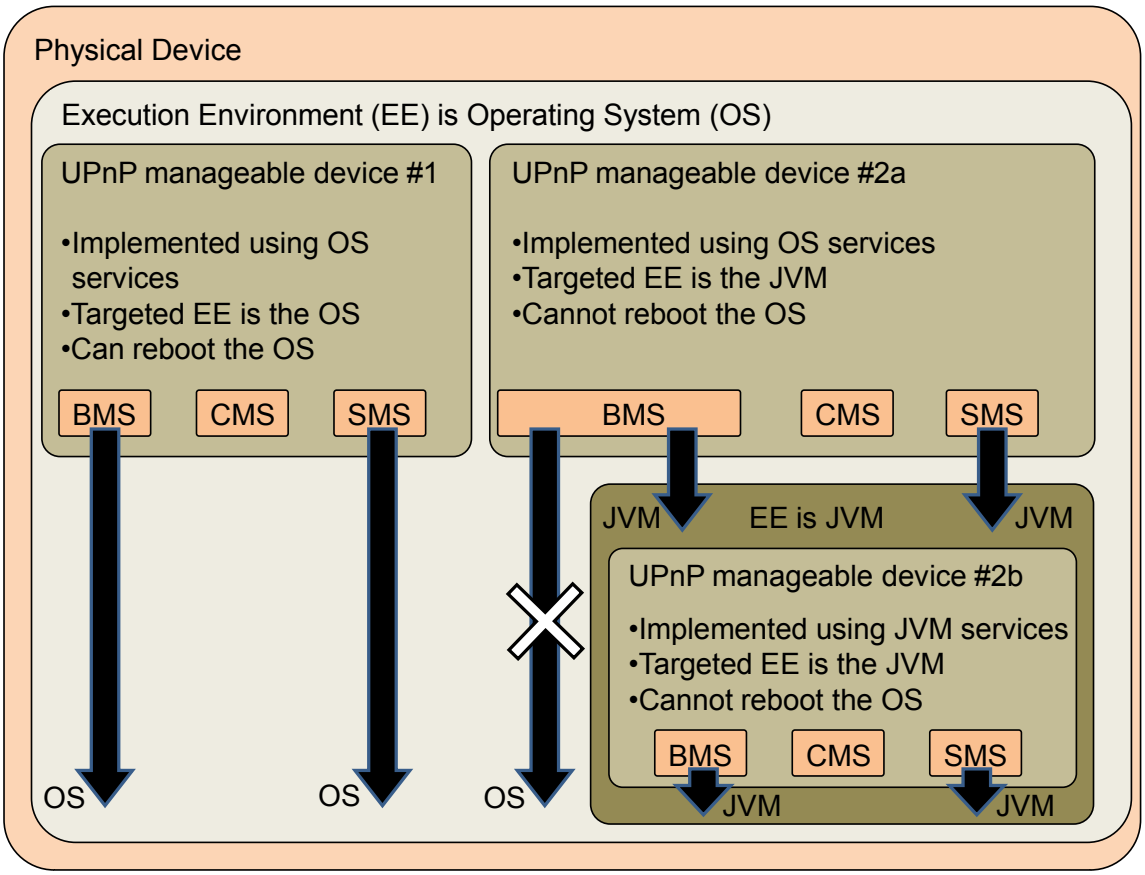
- Names that are well-known or commonly-used abbreviations MAY be used without a prefix.
- Otherwise the protocol name SHOULD begin with a prefix followed by a colon, as specified for *DeviceProtection:1* *Role* names [DPS].

2.7. The f O e a (I f a e)

This non-normative (informative) section walks through several scenarios to illustrate the various actions supported by the *BasicManagement* service.

2.7.1. A

Figure 2-3 illustrates a physical device that hosts two Execution Environments (EE) and three *Parent Devices* (2a and 2b are alternatives).



Parent Devices

2.7.1.1. Parent Device #1

Most examples in this section use the simple *Parent Device* #1:

- It is implemented using Operating System (OS) services.
- The targeted EE is the OS, so any EE-related actions and data model apply to the OS.
- In addition, it is assumed that:

  Both and are " ", so the *Reboot()* and *BaselineReset()* actions apply to the OS. Therefore:

- Rebooting the *Parent Device* involves an OS reboot.
- Resetting the *Parent Device* to its baseline state involves an OS reboot. Persistent settings revert to their baseline values.

- The device implementation is not able to reboot immediately, in which case the action completes successfully and returns a *RebootStatus* value of *RebootLater*. The device will reboot as soon as possible. For example, the device might currently be providing a service such as playing a video, printing a document or hosting a phone call.

On successful completion of a *Reboot()* request, the *Parent Device* *AboutToReboot* internal state is set to "_" and any subsequent action requests are expected to be rejected with a 501 (Action Failed) error code.

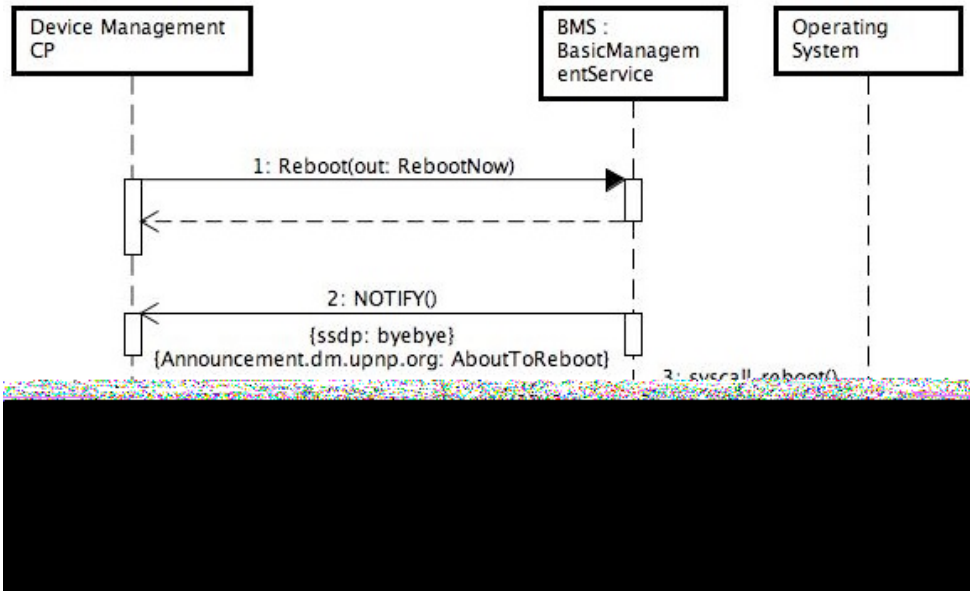
When the *Parent Device* is ready to reboot (which could, if *RebootLater* was returned, be some time later), each _____ message (if sent) will include an *Announcement.dm.upnp.org: AboutToReboot* header. The usual [HTTP] header rules apply, so both of the following are valid:

Announcement.dm.upnp.org: AboutToReboot
ANNOUNCEMENT.DM.UPNP.ORG: AboutToReboot

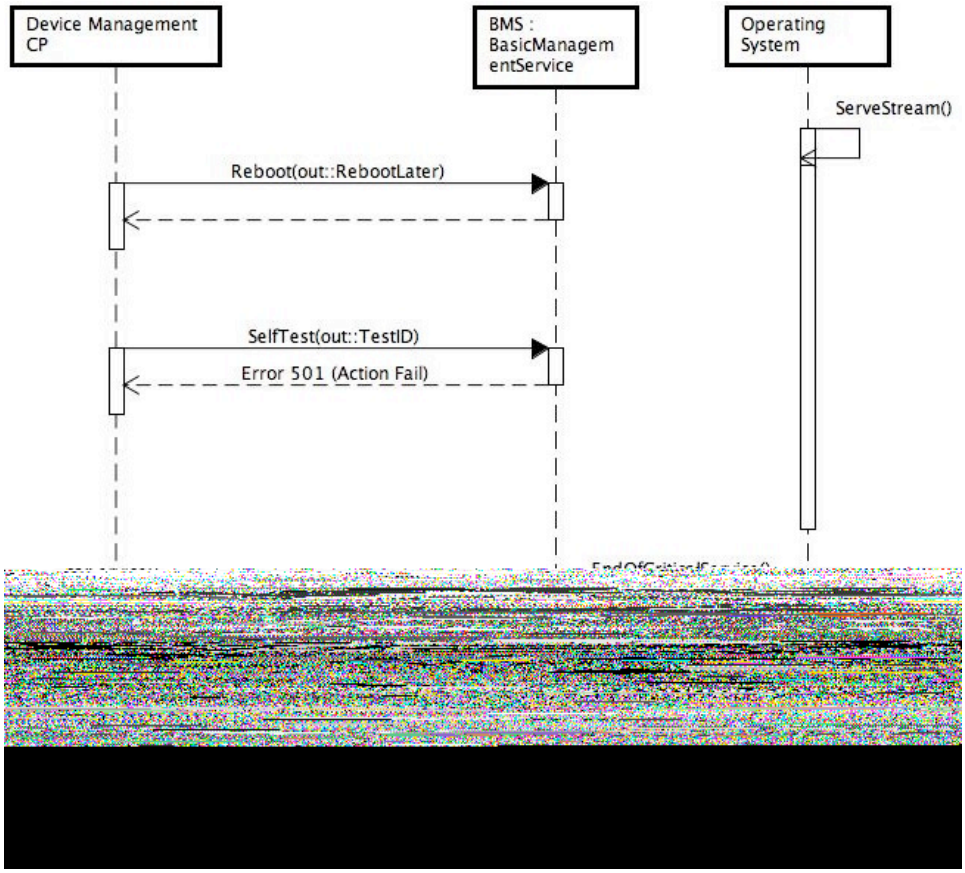
The actual reboot behavior depends on the *Parent Device* configuration:

- For *Parent Device* #1, the OS is rebooted.
- For *Parent Device* #2a, the *Parent Device*, its embedded devices / services, and the targeted EE (JVM) are restarted.
- For *Parent Device* #2b, the targeted EE (JVM) is restarted.

The Figures below illustrate the use of *RebootNow* and *RebootLater*.



RebootNow



RebootLater

2.7.3. Re e g he *Parent Device*

The *BaselineReset()* action (section 2.5.2) is optional. Its *Role List* is "*Admin*" and its *Recommended Role List* is empty. This means that only control points that possess the *Admin Role* are permitted to invoke the action. The action resets the following to their baseline state:

- The UPnP *Parent Device*, including any embedded devices / services.
- The associated device-level entities that are managed via the above devices and services:

  For the *Parent Device*, this means the targeted EE and potentially the OS.

  For any embedded devices, this will depend on the embedded device type.

For a *Parent Device* that has access to the OS, the baseline state is usually referred to as the factory default state. It's up to the implementation to decide exactly what this means.

On successful completion of a *BaselineReset()* request, the *Parent Device AboutToBaselineReset* internal state is set to "_" and any subsequent action requests are expected to be rejected with a 501 (Action Failed) error code. When the device is ready to be reset to its baseline state, each _____ message (if sent) will include an *Announcement.dm.upnp.org: AboutToBaselineReset* header.

In many cases, a baseline reset will involve a reboot. If so, the the *Parent Device AboutToReboot* internal state will be set to "_" and each _____ message (if sent) will include an *Announcement.dm.upnp.-org: AboutToReboot* header. The usual [HTTP] header rules apply, so all of the following are valid:

Announcement.dm.upnp.org: AboutToBaselineReset, AboutToReboot
Announcement.dm.upnp.org: AboutToReboot, AboutToBaselineReset

ANNOUNCEMENT.DM.UPNP.ORG: AboutToReboot, AboutToBaselineReset

ANNOUNCEMENT.DM.UPNP.ORG: AboutToReboot
ANNOUNCEMENT.DM.UPNP.ORG: AboutToBaselineReset

The actual baseline reset behavior depends on the *Parent Device* configuration. In the cases that we are considering here, baseline reset consists of the following:

- Reboot, as described in section 2.7.2.
- Persistent settings revert to their baseline values.

2.7.4. U g Se e ce M de

The *SetSequenceMode()* and *GetSequenceMode()* actions (sections 2.5.4 and 2.5.5) are optional.

- *SetSequenceMode()*'s *Role List* is "*Admin Basic*" and its *Recommended Role List* is "*Public*". This means that control points that possess the *Admin* or *Basic* *Role* are unconditionally permitted to invoke the action. A control point that possesses only the *Public* *Role* is (as explained in section 2.5.4.2) only permitted to invoke the action if its *Identity* is present in the *DeviceProtection:1* ACL.
- *GetSequenceMode()*'s *Role List* is "*Public*". This means that all control points can unconditionally invoke the action. This makes sense, because it simply returns the value of the evented *SequenceMode* state variable.

SetSequenceMode() controls the value of the *SequenceMode* state variable (section 2.3.2). *SequenceMode* can be used to indicate that:

- A control point is planning to execute a sequence of actions.
- A control point is currently executing a sequence of actions.

SequenceMode provides an informal locking mechanism that can affect the behavior of control points and the *Parent Device* implementation. This is not a guaranteed mechanism (the associated requirements are never stronger than "SHOULD") and the *Parent Device* will still behave properly if *SequenceMode* is ignored by all parties. However, if the mechanism is honored then device management can in many cases proceed more efficiently.

The following example assumes that control points A and B both wish to make some configuration changes. Firstly assume that the *Parent Device* can commit and apply each change immediately, without needing to reboot:

- Initially *SequenceMode* is "_".
- Control point A calls *SetSequenceMode("_")* , discovering that it was previously "_", and therefore knowing that it can proceed to make its changes.
- Control point B calls *SetSequenceMode("_")* , discovering that it was previously "_", and therefore knowing that it can't proceed to make its changes.
- Control point A makes its changes, each of which is committed and applied immediately.
- Control point A calls *SetSequenceMode("_")* , indicating that it has finished making its changes.
- Control point B discovers (via polling or eventing) that *SequenceMode* is now "_", so proceeds to set it to "_", make its changes, and set it back to "_".

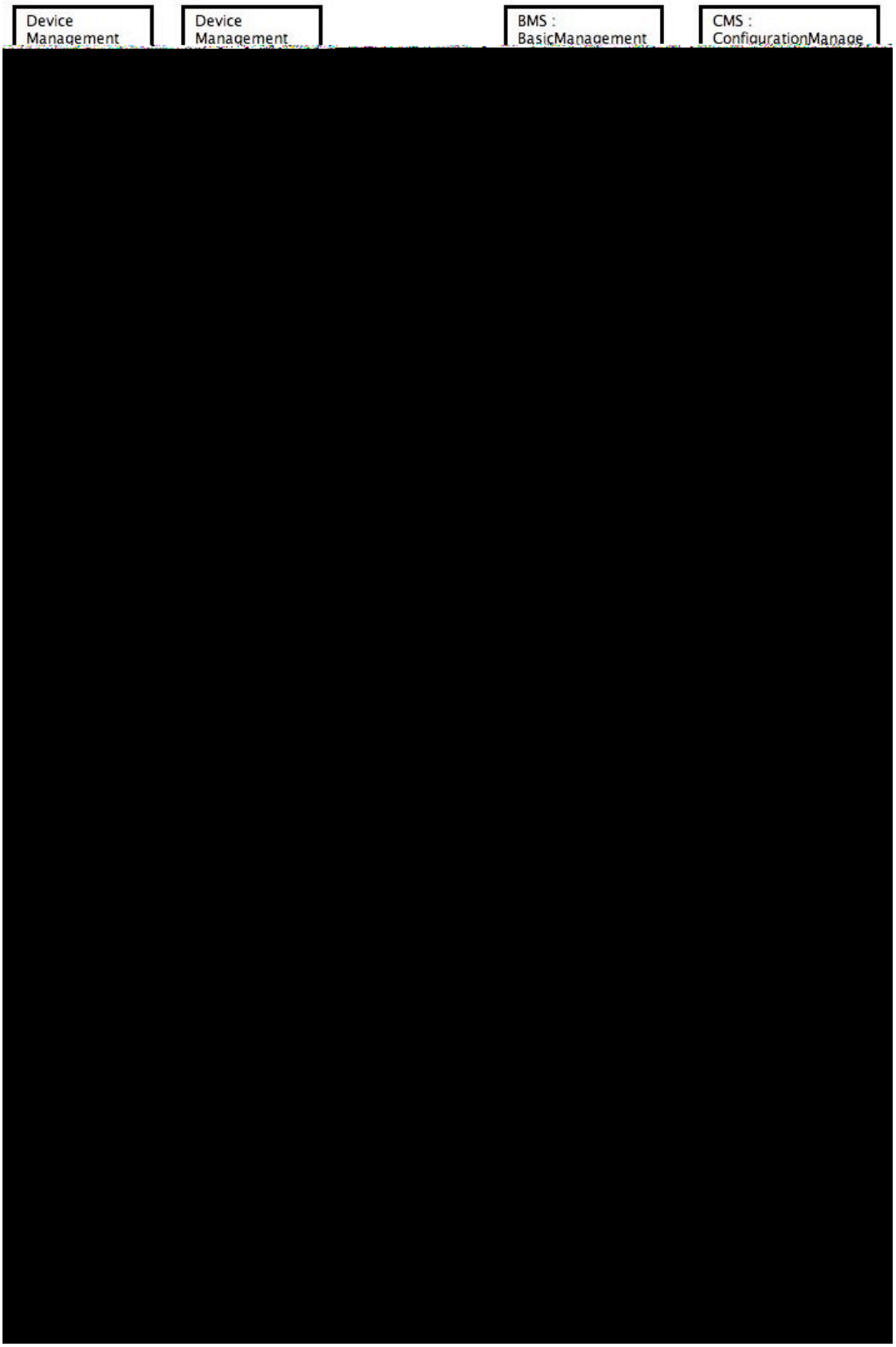
This would clearly work just as well if there had also been a control point C. When control point A had finished its changes and called *SetSequenceMode("_"),* either control point B or control point C would have managed to set *SequenceMode* to "_" and the other one would have had to wait.

In the above example, use of *SequenceMode* was not necessary, because control point A's changes could have been interleaved with control point B's changes, and the end result would have been the same. Indeed for this *Parent Device*, *CMS::SetValues()* can ignore the value of *SequenceMode*.

What if the *Parent Device* needs to reboot in order to apply changes? This doesn't change the control point logic:

- Initially *SequenceMode* is "_".
- Control point A calls *SetSequenceMode("_")* , discovering that it was previously "_", and therefore knowing that it can proceed to make its changes.
- Control point B calls *SetSequenceMode("_")* , discovering that it was previously "_", and therefore knowing that it can't proceed to make its changes.
- Control point A makes its changes. *SequenceMode* is "_", so the *Parent Device* commits changes but doesn't attempt to apply them (which would require a reboot for each change).
- Control point A calls *SetSequenceMode("_")* , indicating that it has finished making its changes. The *Parent Device* now applies the previously-committed changes, resulting in a reboot.
- Control point B discovers (via polling or eventing) that *SequenceMode* is now "_", so proceeds to set it to "_", make its changes, and set it back to "_".

Figure 2-6 below illustrates the use of *SequenceMode*.



SequenceMode

2.7.5. R g a P g Te

The *Ping()* and *GetPingResult()* actions (sections 2.5.6 and 2.5.7) are optional.

- *Ping()*'s and *GetPingResult()*'s *Role Lists* are both "*Public*". This means that all control points can unconditionally invoke the actions.

Suppose that a control point wishes to check that a *Parent Device* can ping *www.myserver.com*. This example illustrates the interactions with the *Parent Device*:

- Control point subscribes to *BasicManagement* events and discovers that *ActiveTestIDs* is "", indicating that no tests are currently active.
- Control point calls *Ping()*. For example:

  Ping("www.upnp.org", 0, 0, 0, 0) : the four zeroes are (respectively) the number of repetitions (defaulted), the timeout (defaulted), the data block size (defaulted) and the DSCP value (best effort). The defaults are implementation-dependent.

  Ping("www.upnp.org", 10, 1000, 32, 16) : 10 repetitions, 1000 millisecond timeout, 32 byte data block, DSCP of 16.
- Control point receives test ID (42 for example) in the *Ping()* response. It also discovers, via an event, that *ActiveTestIDs* is now "42".
- *Parent Device* performs the test and, on completion, removes test ID 42 from *ActiveTestIDs*.
- Control point discovers, via an event, that *ActiveTestIDs* is "" and knows that the test is complete.
- Control point calls *GetPingResult(42)* to retrieve the test results. For example:

  GetPingResult(42) → *("Success", "", 9, 1, 45, 40, 50)* indicates that the ping test was successful, no additional information string was returned, and 9/10 pings succeeded with a mean/min/max response times (for the successful pings) of 45ms/40ms/50ms.

  GetPingResult(42 → *("Error_CannotResolveHostName", "Timeout", 0, 0, 0, 0, 0)* indicates that the ping test failed because the host name couldn't be resolved. The free-format additional info indicates a timeout, and the values of the remaining output arguments are irrelevant (because the test failed).

Alternatively, if the control point doesn't want to use events, once it knows the test ID it can call *GetTestInfo(42)*, which returns the test type (*Ping*) and the test state (*Requested*, *InProgress*, *Canceled*, *Completed*). Once it changes to *Completed*, *GetPingResult(42)* can be called to return the results.

2.7.6. R g a NSL Te

The *NSLookup()* and *GetNSLookupResult()* actions (sections 2.5.8 and 2.5.9) are optional.

- *NSLookup()*'s and *GetNSLookupResult()*'s *Role Lists* are "*Public*". This means that all control points can unconditionally invoke the actions.

Suppose that a control point wishes to check that a *Parent Device* can look up the DNS name *www.myserver.com*. This example illustrates the interactions with the *Parent Device*:

- Control point subscribes to *BasicManagement* events and discovers that *ActiveTestIDs* is "", indicating that no tests are currently active.
- Control point calls *NSLookup()*. For example:

NSLookup("www.myserver.com", "", 0, 0) : the empty string indicates that the default DNS server will be used, and the two zeroes are (respectively) the number of repetitions (defaulted) and the timeout (defaulted). The defaults are implementation-dependent.

NSLookup("www.myserver.com", "mydnsserver.com", 10, 1000) : DNS server *mydnsserver.com*, 10 repetitions, 1000 millisecond timeout.

- Control point receives test ID (43 for example) in the *NSLookup()* response. It also discovers, via an event, that *ActiveTestIDs* is now "43".
- *Parent Device* performs the test and, on completion, removes test ID 43 from *ActiveTestIDs*.
- Control point discovers, via an event, that *ActiveTestIDs* is "" and knows that the test is complete.
- Control point calls *GetNSLookupResult(43)* to retrieve the test results. For example:

  GetNSLookupResult(43)→ ("Success", "", 9, "<?xml...>...") indicates that the DNS lookup test was successful, no additional information string was returned, and 9/10 lookups succeeded, and the detailed results are returned in the XML document (see section 2.3.25 for an example XML document).

  GetNSLookupResult(43) → ("Error_DNSServerNotResolved", "Timeout", 0, "") indicates that the DNS lookup test failed because the DNS server couldn't be resolved. The free-format additional info indicates a timeout, and the values of the remaining output arguments are irrelevant (because the test failed).

Alternatively, if the control point doesn't want to use events, once it knows the test ID it can call *GetTestInfo(43)*, which returns the test type (*NSLookup*) and the test state (*Requested*, *InProgress*, *Canceled*, *Completed*). Once it changes to *Completed*, *GetNSLookupResult(43)* can be called to return the results.

2.7.7. R g a T ace e Te

The *Traceroute()* and *GetTracerouteResult()* actions (sections 2.5.10 and 2.5.11) are optional.

- *Traceroute()*'s and *GetTracerouteResult()*'s *Role Lists* are "*Public*". This means that all control points can unconditionally invoke the actions.

Suppose that a control point wishes to trace the route from a *Parent Device* to *www.myserver.com*. This example illustrates the interactions with the *Parent Device*:

- Control point subscribes to *BasicManagement* events and discovers that *ActiveTestIDs* is "", indicating that no tests are currently active.
- Control point calls *Traceroute()*. For example:

  Traceroute("www.myserver.com", 0, 0, 0, 0) : the four zeroes are (respectively) the timeout (defaulted), the data block size (defaulted), the maximum hop count (defaulted) and the DSCP value (best effort). The defaults are implementation-dependent.

  Traceroute("www.myserver.com", 1000, 32, 20, 16) : 1000 millisecond timeout, 32 byte data block, maximum hop count of 20, DSCP of 16.

- Control point receives test ID (44 for example) in the *Traceroute()* response. It also discovers, via an event, that *ActiveTestIDs* is now "44".
- *Parent Device* performs the test and, on completion, removes test ID 44 from *ActiveTestIDs*.
- Control point discovers, via an event, that *ActiveTestIDs* is "" and knows that the test is complete.

- Control point calls *GetTracerouteResult(44)* to retrieve the test results. For example:

  GetTracerouteResult(44) → *("Success", "", 888, "1.2.3.4,2.3.4.5,,4.5.6.7")* indicates that the trace-route test was successful, no additional information string was returned, the average round-trip time to *www.myserver.com* was 888 milliseconds, and there were four hops to *www.myserver.com*. The third entry in the list of hops is empty, indicating that no replies were received from it. The final entry will be *www.myserver.com*'s IP address.

  GetTracerouteResult(44) → *("Error_MaxHopCountExceeded", "Timeout", 0, "")* indicates that the trace-route test failed because the number of hops to *www.mysever.com* is more than the supplied hop count. The free-format additional info indicates a timeout, and the values of the remaining output arguments are irrelevant (because the test failed).

Alternatively, if the control point doesn't want to use events, once it knows the test ID it can call *GetTestInfo(44)*, which returns the test type (*Traceroute*) and the test state (*Requested*, *InProgress*, *Canceled*, *Completed*). Once it changes to *Completed*, *GetTracerouteResult(44)* can be called to return the results.

2.7.8. R g a l e faceRe e Te

The *InterfaceReset()* and *GetInterfaceResetResult()* actions (sections 2.5.15 and 2.5.16) are optional.

- *InterfaceReset()*'s *Role List* is "*Admin*" and its *Recommended Role List* is "*Basic*". This means that control points that possess the *Admin Role* are unconditionally permitted to invoke the action. A control point that possesses only the *Basic Role* is (as explained in section2.5.15.2) only permitted to invoke the action if the value of the *Interfaces* argument is "*RequestInterface*" (because we are assuming that the *BasicManagement:2* access control list is as listed in the section 2.3.37 example).

- *GetInterfaceResetResult()*'s *Role List* is "*Public*". This means that all control points can unconditionally invoke the action.

Suppose that a control point wishes to reset an IP interface. This example illustrates the interactions with the *Parent Device*:

- Control point subscribes to *BasicManagement* events and discovers that *ActiveTestIDs* is "", indicating that no tests are currently active.

- Control point calls *InterfaceReset()*. For example:

  InterfaceReset("AllInterfaces") : reset all IP interfaces. This is permitted only if the control point possesses the *Admin Role*; for other control points, the action would fail with a 606 (Action not authorized) error.

  InterfaceReset("RequestInterface") : reset the IP interface on which the action request was received. This is permitted only if the control point possesses the *Admin* or *Basic Role*; for other control points, the action would fail with a 606 (Action not authorized) error.

  InterfaceReset("X_UPNP_ORG_lan") : reset the IP interface whose system name is *lan*. If no such interface exists, the request will be rejected with a 701 (Interface Not Found) error. This is permitted only if the control point possesses the *Admin Role*; for other control points, the action would fail with a 606 (Action not authorized) error.

- Control point receives test ID (46 for example) in the *InterfaceReset()* response. It also discovers, via an event, that *ActiveTestIDs* is "46".

- *Parent Device* performs the test. If the UPnP management interface needs to be reset, this will force the *Parent Device* also to be reset, in which case test ID 46 needs to persist across this reset. On completion, test ID 46 is removed from *ActiveTestIDs*.

- Control point discovers, via an event, that *ActiveTestIDs* is "" and knows that the test is complete, i.e., test ID 46 has been removed because of the test completion.

- Control point calls *GetInterfaceResetResult(46)* and retrieves the test results. For example:

  GetInterfaceResetResult(46)→ ("Success", "", 1, 0) indicates that the interface reset test was successful, no additional information string was returned, one IP interface was successfully reset, and no IP interfaces could not be reset.

  GetInterfaceResetResult(46)→ ("Error_Other", "Timeout", 0, 0) indicates that the interface reset test failed. The free-format additional info indicates a timeout, and the values of the remaining output arguments are irrelevant (because the test failed).

Alternatively, if the control point doesn't want to use events, once it knows the test ID it can call *GetTestInfo(46)*, which returns the test type (*InterfaceReset*) and the test state (*Requested*, *InProgress*, *Canceled*, *Completed*). Once it changes to *Completed*, *GetInterfaceResetResult(46)* can be called to return the results.

2.7.9. R g a Se f Te

The *SelfTest()* and *GetSelfTestResult()* actions (sections 2.5.17 and 2.5.18) are optional.

- *SelfTest()*'s *Role List* is "*Admin Basic*" and its *Recommended Role List* is empty. This means that control points that possess the *Admin* or *Basic* *Role* are unconditionally permitted to invoke the action.

- *GetSelfTestResult()*'s *Role List* is "*Public*". This means that all control points can unconditionally invoke the action.

Suppose that a control point wishes to run a self test. This example illustrates the interactions with the *Parent Device*:

- Control point subscribes to *BasicManagement* events and discovers that *ActiveTestIDs* is "", indicating that no tests are currently active.

- Control point calls *SelfTest()*.

- Control point receives test ID (47 for example) in the *SelfTest()* response. It also discovers, via an event, that *ActiveTestIDs* is "47".

- *Parent Device* performs the test and, on completion, removes test ID 47 from *ActiveTestIDs*.

- Control point discovers, via an event, that *ActiveTestIDs* is "" and knows that the test is complete, i.e., test ID 47 has been removed because of the test completion.

- Control point calls *GetSelfTestResult(47)* and retrieves the test results. For example:

  GetSelfTestResult(47)→ (1, "") indicates that the self test was successful, but no additional information string was returned.

  GetSelfTestResult(47)→ (0, "Timeout") indicates that the self test failed. The free-format additional info indicates a timeout.

Alternatively, if the control point doesn't want to use events, once it knows the test ID it can call *GetTestInfo(47)*, which returns the test type (*SelfTest*) and the test state (*Requested*, *InProgress*, *Canceled*, *Completed*). Once it changes to *Completed*, *GetSelfTestResult(47)* can be called to return the results.

2.7.10. Ba d d h Te

This section contains several examples of bandwidth tests. The first example is a basic test that illustrates the general principles. This is followed by illustrations of several use cases associated with the following table:

Tab e 2-77: Ba d d h Te D c e ed De ce

ID	P c	Ve	P f e	E d	D ec
1	*HTTP*	1.1	*Baseline*	*Server*	*Get*
2	*HTTP*	1.1	*Baseline* *BBF*	*Client*	*Get*
3	*Echo*		*Baseline* *BBF*	*Server*	*PutEcho*
4	*Iperf*	2.0.4	*Baseline*	*Server*	*PutEcho*
5	*Iperf*	3.0.0	*Baseline*	*Server*	*PutEcho*
6	*Iperf*	2.0.5	*Baseline*	*Client*	*PutEcho*

Suppose that Table 2-77 is populated by a control point that invokes *GetBandwidthTestInfo()* on all the *Parent Devices* in the home network that support the action. Note that:

- The ID just labels the table rows. Several rows might map to the same *BasicManagement* service instance.
- The *BBF* profile is always a superset of the *Baseline* profile, so if the *BBF* profile is supported, the *Baseline* profile is always supported as well.

We will consider the following four use cases:

- *HTTP Baseline Get*
- *HTTP BBF Get*
- *Echo Baseline*
- *Iperf Baseline*

The control point will use a three-box model (control point, UPnP test server, UPnP test client) if possible, but will use a two-box model (control point with embedded client/server, UPnP test server/client) if necessary.

If the *Parent Device* is a UPnP AV device, one possible deployment is for the UPnP test server to be a *MediaServer* and the UPnP test client to be a *MediaRenderer*. In this case, the bandwidth test can perform a data transfer that is quite similar to a data transfer that plays *MediaServer* content on the *MediaRenderer*.

2.7.10.1. Basic Test

The *GetBandwidthTestInfo()*, *BandwidthTest()* and *GetBandwidthTestResult()* actions (sections 2.5.12, 2.5.13 and 2.5.14) are optional.

- *BandwidthTest()*'s *Role List* is "*Admin Basic*" and its *Recommended Role List* is "*Public*". This means that control points that possess the *Admin* or *Basic* *Role* are unconditionally permitted to invoke the action. A control point that possesses only the *Public* *Role* is (as explained in section 2.5.13.2) only permitted to invoke the action if its *Identity* is present in the *DeviceProtection:1* ACL.

- *GetBandwidthTestInfo()*'s and *GetBandwidthTestResult()*'s *Role Lists* are both "*Public*". This means that all control points can unconditionally invoke these actions.

Suppose that a control point wishes to download a file from *www.myserver.com* to a *Parent Device*. This example illustrates the interactions with the *Parent Device*:

- Control point optionally calls *GetBandwidthTestInfo()* to determine client capabilities. For example:

  GetBandwidthTestInfo() → *"<?xml…> …"* : the client capabilities are returned in the XML document (see section 2.3.27 for an example XML document).

- File server doesn't support *BasicManagement*. So control point has to have prior knowledge of the download URL. We will use *http://www.myserver.com/public/test.dat*.

- Control point subscribes to *BasicManagement* events and discovers that *ActiveTestIDs* is "", indicating that no tests are currently active.

- Control point calls *BandwidthTest()*. For example:

  BandwidthTest("<?xml…> …", "Client", "", "") : the arguments are (respectively) the test specification (see section 2.3.28 for an example), the test endpoint (see section 2.3.16), the test schedule (empty because no test schedule syntax has yet been defined; see section 0), and the session ID (empty because no protocol-specific session ID requirements have yet been defined; see section 2.3.18).

- Control point receives test ID (45 for example) in the *BandwidthTest()* response. It also discovers, via an event, that *ActiveTestIDs* is now "45".

- *Parent Device* performs the test and, on completion, removes test ID 45 from *ActiveTestIDs*.

- Control point discovers, via an event, that *ActiveTestIDs* is "" and knows that the test is complete.

- Control point calls *GetBandwidthTestResult(45)* to retrieve the test results. For example:

  GetBandwidthTestResult(45) → *("Success", "", "<?xml…> …")* indicates that the bandwidth test was successful, no additional information string was returned, and the detailed results are returned in the XML document (see section 2.3.25 for an example XML document)..

  GetBandwidthTestResult(45) → *("Error_InitConnectionFailed", "Timeout", "")* indicates that the bandwidth test failed to connect to *www.mysever.com*. The free-format additional info indicates a timeout, and the values of the other output argument is irrelevant (because the test failed).

Alternatively, if the control point doesn't want to use events, once it knows the test ID it can call *GetTestInfo(45)*, which returns the test type (*Bandwidth*) and the test state (*Requested*, *InProgress*, *Canceled*, *Completed*). Once it changes to *Completed*, *GetBandwidthTestResult(45)* can be called to return the results.

2.7.10.2. HTTP Baseline Get Test

Table 2-77 Row 1 is an *HTTP*/1.1 *Baseline* server and Row 2 is an *HTTP*/1.1 *Baseline* client. The sequence will be:

- Invoke *BandwidthTest()* on the Row 1 service (HTTP server) to prepare the HTTP server for the test.

- Invoke *BandwidthTest()* on the Row 2 service (HTTP client) to initiate the test.

- Wait for the test to complete.

- Invoke *GetBandwidthTestResult()* on the two services.

Here are *BandwidthTestInfo*, *BandwidthTestSpec* and *BandwidthTestResult* XML documents that illustrate this use case.

Row 1 (HTTP server) BandwidthTestInfo

*Row 2 (HTTP client) BandwidthTestInfo (differences from server are shown in **bold**)*

```
<Client>

  <Port/>
```

```
    <TotalBytesReceived/>
  </Client>
```

BandwidthTestSpec (spec is the same for client and server)

Row 1 (HTTP server) BandwidthTestResult

*Row 2 (HTTP client) BandwidthTestResult (differences from server are shown in **bold**)*

```
<Client>
  <TotalBytesReceived>123456</TotalBytesReceived>
</Client>
```

2.7.10.3. HTTP BBF Get Test

Table 2-77 Row 2 is an *HTTP*/1.1 *BBF* client but there is no *HTTP* *BBF* server, so the control point will use its own embedded HTTP server (an alternative would be to use a well-known external server). The sequence will be:

- Prepare the embedded HTTP server for the test.
- Invoke *BandwidthTest()* on the Row 2 service (HTTP client) to initiate the test.
- Wait for the test to complete.
- Invoke *GetBandwidthTestResult()* on the Row 2 service.

Here are *BandwidthTestInfo*, *BandwidthTestSpec* and *BandwidthTestResult* XML documents that illustrate the use case.

*Row 2 (HTTP client) BandwidthTestInfo (differences from Baseline are shown in **bold**)*

```
                                                    BBF
                                                BaseProfiles="Baseline"
```

```
<DSCP/>
<EthernetPriority/>
<ROMTime/>
<BOMTime/>
<EOMTime/>
```

```
<TestBytesReceived/>
```

*BandwidthTestSpec (differences from Baseline are shown in **bold**)*

```
                                                    BBF
                                                BaseProfiles="Baseline"
```

*Row 2 (HTTP client) BandwidthTestResult (differences from Baseline are shown in **bold**)*

```
                                                    BBF
                                                BaseProfiles="Baseline"
```

```
<DSCP>0</DSCP>
<EthernetPriority>0</EthernetPriority>

<ROMTime>2008-04-09T15:01:05.123456</ROMTime>
<BOMTime>2008-04-09T15:01:05.123456</BOMTime>
<EOMTime>2008-04-09T15:01:05.123456</EOMTime>
<TestBytesReceived>123456</TestBytesReceived>
```

2.7.10.4. Echo Baseline Test

Table 2-77 Row 3 is an *Echo Baseline* server but there is no *Echo* client, so the control point will use its own embedded Echo client (an alternative would be to use a well-known external client). The sequence will be:

- Invoke *BandwidthTest()* on the Row 3 service (Echo server) to prepare the Echo server for the test.

- Enable the embedded Echo client.
- Optionally invoke *GetBandwidthTestResult()* on the Row 3 service (Echo server) to return intermediate results.
- Wait for the test to complete (or cancel the test when it is no longer needed).
- Optionally invoke *GetBandwidthTestResult()* on the Row 3 service (Echo server) to return the final results.

Here are *BandwidthTestInfo*, *BandwidthTestSpec* and *BandwidthTestResult* XML documents that illustrate the use case.

Row 3 (Echo server) BandwidthTestInfo

BandwidthTestSpec

Row 3 (Echo server) BandwidthTestResult

2.7.10.5. Iperf Baseline Test

Table 2-77 Row 4 is an *Iperf* v2.0.4 server and Row 6 is an *Iperf* v2.0.5 client. These are both Iperf v2 and so can interoperate (Row 5 is an *Iperf* v3.0.0 server which cannot interoperate with the Row 6 client). The sequence will be:

- Invoke *BandwidthTest() on* the Row 4 service (Iperf server) to prepare the Iperf server for the test.
- Invoke *BandwidthTest()* on the Row 6 service (Iperf client) to initiate the test.
- Wait for the test to complete.
- Invoke *GetBandwidthTestResult()* on the two services.

Here are *BandwidthTestInfo*, *BandwidthTestSpec* and *BandwidthTestResult* XML documents that illustrate this use case.

Row 4 (Iperf server) BandwidthTestInfo

*Row 6 (Iperf client) BandwidthTestInfo (differences from server are shown in **bold**)*

```
                                    2.0.5
<Client>




      <AllowedValue>5002</AllowedValue>



  <Direction>
    <AllowedValueList>
      <AllowedValue>Put</AllowedValue>
      <AllowedValue>PutEcho</AllowedValue>
    </AllowedValueList>
  </Direction>
  <TimeMSecs/>
  <LengthBytes/>
  <BandwidthBytesPerSec/>









</Client>
```

BandwidthTestSpec (note that Version="2" matches the server and client versions)

Row 4 (Iperf server) BandwidthTestResult

*Row 6 (Iperf client) BandwidthTestResult (differences from server are shown in **bold**)*

```
<Client>

</Client>
```

2.7.11. Ma a g L g

The *GetLogURIs()* and *GetLogInfo()* actions (sections 2.5.23 and 2.5.25) are optional. Their *Role Lists* are both "*Public*". This means that all control points can unconditionally invoke the actions.

The *SetLogInfo()* action (section 2.5.24) is optional. Its *Role List* is "*Admin*" and its *Recommended Role List* is "*Basic*". This means that control points that possess the *Admin Role* are unconditionally permitted to invoke the action. A control point that possesses only the *Basic Role* is (as explained in section 2.5.24.2) only permitted to invoke the action if the value of the *LogURI* argument contains the string ":restricted:" (because we are assuming that the *BasicManagement:2* access control list is as listed in the section 2.3.37 example).

The evented *LogURIs* state variable contains a list of the URIs of each log that is currently supported by the *Parent Device*. This list can also be retrieved via the *GetLogURIs()* action. For example:

- *GetLogURIs() → ("urn:example-com:private:device-log")* : a single log for *Parent Device* #1 and #2a.
- *GetLogURIs() → ("urn:example-com:private:device-log,urn:example-com:restricted:jvm-log")* : two logs for *Parent Device* #2b.

Each log URI uniquely identifies a log and is used as an argument to the remaining log-related actions. For example:

- *GetLogInfo("urn:example-com:private:device-log") → (1, 1, "Error", "http://192.168.1.254/device-log", 0, 2009-06-15T14:00:00)* indicates that the specified log is configurable, is enabled, its current log level (*Error*), its log URL (*http://192.168.1.254/device-log*), its maximum size (0 means unknown), and the time at which it last changed.
- *GetLogInfo("urn:example-com:restricted:jvm-log") → (0, 1, "Info", "http://192.168.1.254/jvm-log", 100000, 2009-06-15T14:00:00)* indicates that the specified log is not configurable, is enabled, its current log level (*Info*), its log URL (*http://192.168.1.254/jvm-log*), its maximum size (100000 bytes), and the time at which it last changed.

- *SetLogInfo("urn:example-com:private:device-log", 1, Info)* changes the log level from *Error* to *Info*. It is permitted only for control points that possess the *Admin Role*; for other control points, the action would fail with a 606 (Action not authorized) error.

- *SetLogInfo("urn:example-com:restricted:jvm-log", 1, Info)* fails with a 711 (Log Not Configurable) error. It is permitted only for control points that possess the *Admin* or *Basic Roles*; for a control point that possessed only the *Public Role*, the action would fail with a 606 (Action not authorized) error.

3. XML Service Description (Normative)

4. XML Schema (Normative)

5. Ve H (I f a e)

BasicManagement:1

- Original

BasicManagement:2

- Added *Diagnostics Feature*: new *GetTestIDs()* and existing *GetActiveTestIDs()*, *GetTestInfo()* and *CancelTest()* actions.
- Added *Ping Diagnostics* etc features (one for each diagnostic test): each such feature includes the requirements of the *Diagnostics Feature*, and also the actions that are specific to the test, e.g. *Ping()* and *GetPingResult()*.
- Added *Security Feature*: *GetACLData()* action.
- Added bandwidth tests: *GetBandwidthInfo()*, *BandwidthTest()* and *GetBandwidthTestResult()* actions.