BasicManagement:2
Service Template Version 1.01
For UPnP Version 1.0
Status: Standardized DCP (SDCP)
Date: 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.

THE UPnP FORUM TAKES NO POSITION AS TO WHETHER ANY INTELLECTUAL PROPERTY RIGHTS EXIST IN THE STANDARDIZED DCPS. THE STANDARDIZED DCPS ARE PROVIDED "AS IS" AND "WITH ALL FAULTS". THE UPnP FORUM MAKES NO WARRANTIES, EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE STANDARDIZED DCPS, INCLUDING BUT NOT LIMITED TO ALL IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE, OF REASONABLE CARE OR WORKMANLIKE EFFORT, OR RESULTS OR OF LACK OF NEGLIGENCE.

© 2012 UPnP Forum. All Rights Reserved.

<table>
<thead>
<tr>
<th>Authors</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>William Lupton (Editor)</td>
<td>2Wire / Pace</td>
</tr>
<tr>
<td>Davide Moreo</td>
<td>Telecom Italia</td>
</tr>
<tr>
<td>Francois-Gaël Ottogalli</td>
<td>France Telecom Group</td>
</tr>
<tr>
<td>Xavier Roubaud</td>
<td>France Telecom Group</td>
</tr>
<tr>
<td>Wouter van der Beek</td>
<td>Philips</td>
</tr>
<tr>
<td>Kiran Vedula</td>
<td>Samsung Electronics</td>
</tr>
</tbody>
</table>

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.
Contents

1. OVERVIEW AND SCOPE ....................................................................................................................... 7
   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. SERVICE MODELING DEFINITIONS ................................................................................................. 18
   2.1. SERVICE TYPE ......................................................................................................................... 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_BandwidthTestId ....................................................................................... 40
       2.3.30. A_ARG_TYPE_BandwidthTestStatus .................................................................................. 40

Copyright UPnP Forum © 2012. All rights reserved.
2.3.1.  A_ARG_TYPE_Interfaces .......................................................... 43
2.3.2.  A_ARG_TYPE_InterfaceResetStatus ........................................ 44
2.3.3.  A_ARG_TYPE_LogURI .................................................................. 44
2.3.4.  A_ARG_TYPE_LogURL ................................................................. 44
2.3.5.  A_ARG_TYPE_LogLevel ............................................................... 45
2.3.6.  A_ARG_TYPE_LogMaxSize ......................................................... 45
2.3.7.  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
3. XML SERVICE DESCRIPTION (NORMATIVE) ........................................................................ 116
4. XML SCHEMA (NORMATIVE) .................................................................................................. 131
5. VERSION HISTORY (INFORMATIVE) ......................................................................................... 145

List of Tables
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 .......................................................... 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

Copyright UPnP Forum © 2012. All rights reserved.
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 GetTestIDs() ............................................................................................................................................. 70
Table 2-53: Arguments for GetActiveTestIDs() ................................................................................................................................. 70
Table 2-54: Error Codes for GetActiveTestIDs() ................................................................................................................................. 71
Table 2-55: Arguments for GetTestInfo() ............................................................................................................................................. 71
List of Figures

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

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 SetLogLevel() .................................................................... 73
Table 2-62: Error Codes for SetLogLevel() .................................................................. 74
Table 2-63: Arguments for GetLogLevel() .................................................................... 74
Table 2-64: Error Codes for GetLogLevel() .................................................................. 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
1. **Overview and Scope**

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. **Introduction**

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 ssdp:alive messages.
- “Parent Device stop” refers to Parent Device shutdown, including (if possible) the sending out of ssdp:byebye 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. **References**

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:


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


Copyright UPnP Forum © 2012. All rights reserved.


1.3. Glossary

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. Notation

- 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.

Copyright UPnP Forum © 2012. All rights reserved.
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 data 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 arch 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. Data Types

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 “0” for false, and the value “1” for true. However, when used as input arguments, the values “false”,
“no”, “true”, “yes” may also be encountered and MUST be accepted. Nevertheless, it is strongly
RECOMMENDED that all state variables and output arguments be represented as “0” and “1”.

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

1.4.2. Strings Embedded in Other Strings

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 “&amp;” or via a numeric character reference.

b) Left angle bracket (“<”) is represented as “&lt;” 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 “&gt;” or via a numeric character reference.

1.5. Derived Data Types

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 string data type is used to define state variable and action argument string data types. The XML Schema namespace is used to define xsd:string data types. The following definition applies to both string data types.

1.5.1. Comma Separated Value (CSV) Lists

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 string or xsd:string and denoted by CSV (x), where x is the type of the individual values.

- The data type of a heterogeneous list is also string or xsd:string 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 string or xsd:string 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 A_ARG_TYPE data types for string lists, and as [XML-SCHEMA-2] data types for xsd:string lists.

- A list is represented as a string type (for state variables and action arguments) or xsd:string 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 “0” for false or “1” for true. These values are a subset of the defined Boolean data type values specified in [UDA1.0]: 0, false, no, 1, true, yes.
• Boolean values are represented in other XML element CSVs as either “0” for false or “1” for true. These values are a subset of the defined Boolean data type values specified in [XML-SCHEMA-2]: 0, false, 1, true.

• 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.

**Table 1-1: CSV Examples**

<table>
<thead>
<tr>
<th>Type refinement of string</th>
<th>Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSV (string)</td>
<td>“first,second”</td>
<td>List of 2 strings used as state variable or action argument value.</td>
</tr>
<tr>
<td>CSV (xsd:string)</td>
<td>“first,second”</td>
<td>List of 2 strings used within an XML element</td>
</tr>
<tr>
<td>CSV (xsd:token)</td>
<td>“first, second ”</td>
<td>List of 2 strings used within an XML element. Each element is of type xsd:token so, even though the second value is “second” and has leading and trailing spaces, the value seen by the application will be “second” because xsd:token collapses whitespace.</td>
</tr>
<tr>
<td>CSV (string, date-Time.tz [, string])</td>
<td>“Warning,2009-07-07T13:22:41, third&quot;,value”</td>
<td>List of string, dateTime.tz and (optional) string used as state variable or action argument value. Note the leading space and escaped comma in the third value, which is “third,value”.</td>
</tr>
<tr>
<td>CSV (string, date-Time.tz [, string])</td>
<td>“Warning,2009-07-07T13:22:41,”</td>
<td>As above but third value is empty.</td>
</tr>
<tr>
<td>CSV (string, date-Time.tz [, string])</td>
<td>“Warning,2009-07-07T13:22:41”</td>
<td>As above but third value is omitted.</td>
</tr>
<tr>
<td>CSV (A_ARG_TYPE_HOST)</td>
<td>“grumpy,sleepy”</td>
<td>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.</td>
</tr>
<tr>
<td>CSV (i4)</td>
<td>“1, 2”</td>
<td>Illegal CSV. White space is not allowed as part of an integer value.</td>
</tr>
<tr>
<td>CSV (string)</td>
<td>“a,,c,,”</td>
<td>List of 4 strings “a”, “”, “c” and “”.</td>
</tr>
</tbody>
</table>
1.5.2. Embedded XML Documents

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 <SupportedDataModels …> root element, optionally preceded by the XML declaration <?xml version="1.0" ...?>.

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

“<SupportedDataModels …>...</SupportedDataModels>”

or

“<?xml ...?><SupportedDataModels ...>...</SupportedDataModels>”

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. Management of XML Namespaces in Standardized DCPs

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

<table>
<thead>
<tr>
<th>Standard Name-space Prefix</th>
<th>Namespace Name</th>
<th>Namespace Description</th>
<th>Normative Definition Document Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DM Working Committee defined namespaces</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bms</td>
<td>urn:schemas-upnp-org:dm:bms</td>
<td>BMS data structures</td>
<td>[BMS]</td>
</tr>
<tr>
<td>cms</td>
<td>urn:schemas-upnp-org:dm:cms</td>
<td>CMS data structures</td>
<td>[CMS]</td>
</tr>
<tr>
<td>sms</td>
<td>urn:schemas-upnp-org:dm:sms</td>
<td>SMS data structures</td>
<td>[SMS]</td>
</tr>
<tr>
<td>bmsnsl¹</td>
<td>urn:schemas-upnp-org:dm:bms:nsl</td>
<td>BMS NSLookupResult</td>
<td>[BMS]</td>
</tr>
<tr>
<td></td>
<td>Externally defined namespaces</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xsd</td>
<td><a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a></td>
<td>XML Schema Language 1.0</td>
<td>[XML-SCHEMA-1]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[XML-SCHEMA-2]</td>
</tr>
<tr>
<td>xsi</td>
<td><a href="http://www.w3.org/2001/XMLSchema-instance">http://www.w3.org/2001/XMLSchema-instance</a></td>
<td>XML Schema Instance Document schema</td>
<td>Sections 2.6 &amp; 3.2.7 of [XML-SCHEMA-1]</td>
</tr>
</tbody>
</table>

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

Table 1-3: Schema-related Information

Copyright UPnP Forum © 2012. All rights reserved.
## Namespace Names, Namespace Versioning and Schema Versioning

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 `xmlns` attribute within the root element. Each `xmlns` 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 `schemaLocation` attribute also within the root element. (See Section 1.6.2)

---

1. **Absolute URIs are generated by prefixing the relative URIs with “http://www.upnp.org/schemas/dm/”.

2. **bmsnsl** was defined in **BasicManagement:1** and remains valid. Its definitions are also present in the **bms** namespace, which is where any future enhancements will be made.
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 "-" yyyyymmdd ".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.
• yyyyymmdd 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 "cms" 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 “cms” 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 “cms” 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 “cms” 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 “cms” 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 schemaLocation 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 schemaLocation 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 version attribute in the <schema> root element. Its value MUST correspond to the format:

```
ver="" yyyyMMdd where ver and yyyyMMdd are described above.
```

The version attribute provides self-identification of the namespace version and release date of the schema itself. For example, within the original schema released for the “cms” namespace (…/cms-v1-20091231.xsd), the <schema> root element might contain the following attribute: version="1-20091231".

### 1.6.2. Namespace Usage Examples

The schemaLocation 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 schemaLocation 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.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<cms:ParameterValueList
 xmlns:cms="urn:schemas-upnp-org:dm:cms"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:schemas-upnp-org:dm:cms:
 http://www.upnp.org/schemas/dm/cms.xsd">
  <Parameter>
    <Path>...</Path>
    <Value>...</Value>
  </Parameter>
  ...
</cms:ParameterValueList>
```

### 1.7. Vendor-defined Extensions

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”.

Copyright UPnP Forum © 2012. All rights reserved.
2. **Service Modeling Definitions**

2.1. **ServiceType**

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

```
urn:schemas-upnp-org:service:BasicManagement:2
```

2.2. **Key Concepts**

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. **Maintenance**

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. **Diagnostic tests**

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.

Figure 2-1: Bandwidth Test Overview

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 $\text{BandwidthTest()}$ 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 $\text{GetBandwidthTestResult()}$ on the server to get server-side test result.

9. Control point calls $\text{GetBandwidthTestResult()}$ on the client to get client-side test result.

If the server doesn’t support $\text{BasicManagement}$ (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 $\text{BasicManagement}$ point of view one would see the following.

1. Control point calls $\text{GetBandwidthTestInfo()}$ 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 $\text{BandwidthTest()}$ 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 $\text{GetBandwidthTestResult()}$ on the client to get client-side test result.

### 2.2.3. Logging

A $\text{Parent Device}$ 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 $\text{Parent Device}$ restart. Log contents and URL MAY persist across a $\text{Parent Device}$ 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 $\text{Parent Device}$’s default (or primary) log.

Unless otherwise specified by a UPnP working committee, UPnP actions invoked on a $\text{Parent Device}$ 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. Security

Actions that do not return sensitive information, change the device configuration, or affect normal device operation are referred to as $\text{Non-Restrictable}$ 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. BasicManagement:2 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:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bms:ACL xmlns:bms="urn:schemas-upnp-org:dm:bms"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:schemas-upnp-org:dm:bms
http://www.upnp.org/schemas/dm/bms.xsd">
  <ACLEntry>
    <Action>InterfaceReset</Action>
    <Filter>Interfaces="RequestInterface"</Filter>
    <Roles>Basic</Roles>
  </ACLEntry>
  <ACLEntry>
    <Action>SetLogInfo</Action>
    <Filter>contains(LogURI,:restricted:)"</Filter>
  </ACLEntry>
</ACL>
```

Copyright UPnP Forum © 2012. All rights reserved.
...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. State Variables

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

**Table 2-1: State Variables**

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Req. or Opt.¹</th>
<th>Data Type</th>
<th>Allowed Value</th>
<th>Default Value</th>
<th>Eng. Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeviceStatus</td>
<td>R</td>
<td>string</td>
<td>CSV² (string, dateTime, tz [, string]), where first string is: OK, Warning, Error, Fatal (section 2.3.1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SequenceMode</td>
<td>O</td>
<td>boolean</td>
<td>(section 2.3.2)</td>
<td>“0”</td>
<td></td>
</tr>
<tr>
<td>TestIDs</td>
<td>CR²</td>
<td>string</td>
<td>CSV (A ARG-TYPE TestID) (section 2.3.3)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright UPnP Forum © 2012. All rights reserved.
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Req. or Opt.</th>
<th>Data Type</th>
<th>Allowed Value</th>
<th>Default Value</th>
<th>Eng. Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveTestIDs</td>
<td>CR¹</td>
<td>string</td>
<td>CSV (A_ARG_TYPE TestID)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LogURIs</td>
<td>O</td>
<td>string</td>
<td>CSV (A_ARG_TYPE Log-URI)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_Boolean</td>
<td>O</td>
<td>boolean</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_String</td>
<td>O</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_UShort</td>
<td>O</td>
<td>ui2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_UInt</td>
<td>O</td>
<td>ui4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_DateTime</td>
<td>O</td>
<td>dateTime</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_MSecs</td>
<td>O</td>
<td>ui4</td>
<td></td>
<td>Milliseconds</td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_RebootStatus</td>
<td>O</td>
<td>string</td>
<td>RebootNow, RebootLater</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_TestID</td>
<td>O</td>
<td>ui4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_TestType</td>
<td>O</td>
<td>string</td>
<td>BandwidthTest, InterfaceReset, NSLookup, Ping, SelfTest, Traceroute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_TestState</td>
<td>O</td>
<td>string</td>
<td>Requested, InProgress, Canceled, Completed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_TestEndpoint</td>
<td>O</td>
<td>string</td>
<td>Server, Client</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_TestSchedule</td>
<td>O</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_TestSessID</td>
<td>O</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_DSCP</td>
<td>O</td>
<td>ui1</td>
<td>Between 0 and 63 inclusive</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_Host</td>
<td>O</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variable Name</td>
<td>Req. or Opt.</td>
<td>Data Type</td>
<td>Allowed Value</td>
<td>Default Value</td>
<td>Eng. Units</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------</td>
<td>-----------</td>
<td>-------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>------------</td>
</tr>
<tr>
<td>A_ARG_TYPE_Hosts</td>
<td>O</td>
<td>string</td>
<td>CSV (A_ARG_TYPE_Host)</td>
<td>Default</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(section 2.3.21)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_HostName</td>
<td>O</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(section 2.3.22)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_PingStatus</td>
<td>O</td>
<td>string</td>
<td>Success, Error CannotResolveHostName</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(section 2.3.23)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_NSLookupStatus</td>
<td>O</td>
<td>string</td>
<td>Success, Error DNS ServerNotResolved</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(section 2.3.24)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_NSLookupResult</td>
<td>O</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(section 2.3.25)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_TracerouteStatus</td>
<td>O</td>
<td>string</td>
<td>Success, Error CannotResolveHostName, Error MaxHopCountExceeded</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(section 2.3.26)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_BandwidthTestInfo</td>
<td>O</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(section 2.3.27)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_BandwidthTestSpec</td>
<td>O</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(section 2.3.28)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_BandwidthTestStatus</td>
<td>O</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(section 2.3.29)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_BandwidthTestResult</td>
<td>O</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(section 2.3.30)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_Interfaces</td>
<td>O</td>
<td>string</td>
<td>AllInterfaces, RequestInterface</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(section 2.3.31)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_InterfaceResetStatus</td>
<td>O</td>
<td>string</td>
<td>Success, Error</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(section 2.3.32)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_LogURI</td>
<td>O</td>
<td>uri</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(section 2.3.33)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_LogURL</td>
<td>O</td>
<td>uri</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(section 2.3.34)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_ARG_TYPE_LogLevel</td>
<td>O</td>
<td>string</td>
<td>Emergency, Alert, Critical, Error, Warning, Notice, Informational, Debug</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(section 2.3.35)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variable Name</td>
<td>Req. or Opt.</td>
<td>Data Type</td>
<td>Allowed Value</td>
<td>Default Value</td>
<td>Eng. Units</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-------------</td>
<td>-----------</td>
<td>---------------</td>
<td>---------------</td>
<td>-----------</td>
</tr>
<tr>
<td>A_ARG_TYPE_LogMaxSize</td>
<td>O</td>
<td>ui4</td>
<td>(section 2.3.36)</td>
<td>Default</td>
<td>Bytes</td>
</tr>
<tr>
<td>A_ARG_TYPE_ACL</td>
<td>CR&lt;sup&gt;4&lt;/sup&gt;</td>
<td>string</td>
<td>(section 2.3.37)</td>
<td>TBD</td>
<td>TBD</td>
</tr>
</tbody>
</table>

Non-standard state variables implemented by an UPnP vendor go here.

<sup>1</sup> R = REQUIRED, O = OPTIONAL, CR = CONDITIONALLY REQUIRED, X = Non-standard.

<sup>2</sup> 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).

<sup>3</sup> REQUIRED if the Diagnostics Feature is supported.

<sup>4</sup> 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 (string, dateTime [ , string]) 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 “0” → “1” transition (via SetSequenceMode(“1”)):

- 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(“1”) or by invoking any action whose
behavior can be affected by `SequenceMode`. When the timer expires, `SequenceMode` is automatically set to “0”.

- 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 “1” when the action is invoked.

A `SequenceMode “1” → “0” transition` (either via `SetSequenceMode(“0”)` or via timer expiry as described above):

- Indicates that all the actions of a sequence executed while `SequenceMode` was “1” have now been invoked.

- Requests that the `Parent Device` MUST as soon as possible apply any changes that were not applied while `SequenceMode` was “1”, and complete any operations that were not performed while `SequenceMode` was “1”.

A `SequenceMode` value of “1” 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:

- **Configure parameters**: 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 = “1”` 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.

- **Install deployment unit**: disables UPnP control interface before downloading file, then reboots.
  - `SequenceMode = “1”` 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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Req. or Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RebootNow</td>
<td>R</td>
<td>The <em>Reboot()</em> request was accepted, nothing in the current <em>Parent Device</em> state precludes an immediate reboot, and the <em>Parent Device</em> will initiate the reboot immediately.</td>
</tr>
</tbody>
</table>
### 2.3.13. **A_ARG_TYPE_TestID**

A test ID. A new test ID is allocated each time a test is successfully requested. This ID is returned to the control point in the action response. Once a test ID has been allocated, the same test ID MUST NOT be reused until the next time the *Parent Device* restarts. Test IDs MAY persist across such *Parent Device* restarts. Individual tests MAY state more stringent test ID retention requirements. The test state transition diagram is shown in Figure 2-2.

### 2.3.14. **A_ARG_TYPE_TestType**

Identifies the type of a given test, e.g. the test that is associated with a specified test ID. Allowed values are listed in Table 2-3.

#### Table 2-3: allowedValueList for **A_ARG_TYPE_TestType**

<table>
<thead>
<tr>
<th>Value</th>
<th>Req. or Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BandwidthTest</td>
<td>CR&lt;sup&gt;1&lt;/sup&gt;</td>
<td></td>
</tr>
<tr>
<td>InterfaceReset</td>
<td>CR&lt;sup&gt;1&lt;/sup&gt;</td>
<td></td>
</tr>
<tr>
<td>NSLookup</td>
<td>CR&lt;sup&gt;1&lt;/sup&gt;</td>
<td></td>
</tr>
<tr>
<td>Ping</td>
<td>CR&lt;sup&gt;1&lt;/sup&gt;</td>
<td></td>
</tr>
<tr>
<td>SelfTest</td>
<td>CR&lt;sup&gt;1&lt;/sup&gt;</td>
<td></td>
</tr>
<tr>
<td>Traceroute</td>
<td>CR&lt;sup&gt;1&lt;/sup&gt;</td>
<td></td>
</tr>
<tr>
<td>Vendor-defined</td>
<td>X&lt;sup&gt;2&lt;/sup&gt;</td>
<td></td>
</tr>
</tbody>
</table>

<sup>1</sup> REQUIRED if the *Diagnostics Feature* and the corresponding diagnostic test actions are supported.

<sup>2</sup> For every vendor-defined diagnostic test, there MUST be a corresponding **A_ARG_TYPE_TestType** allowed value.

### 2.3.15. **A_ARG_TYPE_TestState**

Indicates the state of a given test, e.g. the test that is associated with a specified test ID. Allowed values are listed in Table 2-4. The test state transition diagram is shown in Figure 2-2.

#### Table 2-4: allowedValueList for **A_ARG_TYPE_TestState**

<table>
<thead>
<tr>
<th>Value</th>
<th>Req. or Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requested</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>InProgress</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Canceled</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Completed</td>
<td>R</td>
<td></td>
</tr>
</tbody>
</table>

Copyright UPnP Forum © 2012. All rights reserved.
2.3.16. **A_ARG_TYPE_TestEndpoint**
Indicates whether a participant in a bandwidth test should play the role of server or of client. Allowed values are listed in Table 2-5.

Table 2-5: allowedValueList for **A_ARG_TYPE_TestEndpoint**

<table>
<thead>
<tr>
<th>Value</th>
<th>Req. or Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>O¹</td>
<td></td>
</tr>
<tr>
<td>Client</td>
<td>O¹</td>
<td></td>
</tr>
</tbody>
</table>

¹ At least one of the values MUST be supported, as indicated by the bandwidth test capabilities returned by `GetBandwidthTestInfo()` (section 2.5.12).

2.3.17. **A_ARG_TYPE_TestSchedule**
An XML document that specifies when a given test will be run, whether it will automatically be repeated, etc. An empty string indicates the service’s default test scheduling policy of “as soon as possible”, as described in section 2.2.2.

This version of the specification does not define an XML Schema for this state variable. The state variable is currently just a placeholder.

2.3.18. **A_ARG_TYPE_TestSessID**
A session ID, chosen by the control point, that is sent first to the server, and then to the client, when initiating a bandwidth test.

This version of the specification does not define any protocol-specific requirements for how this session ID should be used. The state variable is currently just a placeholder.

2.3.19. **A_ARG_TYPE_DSCP**
The DiffServ Code Point [DSCP] to be used in packets that are sent during the execution of a test. The value MUST be in the range 0 to 63.

2.3.20. **A_ARG_TYPE_Host**
An IPv4/IPv6 address, DNS name, or empty string.
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>
<thead>
<tr>
<th>Value</th>
<th>Req. or Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Error_CannotResolveHostName</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Error_Internal</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>Error_Other</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>Vendor-defined</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

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_NSLookupResult Status, which indicates the success or failure of a given repetition of the test.

<table>
<thead>
<tr>
<th>Value</th>
<th>Req. or Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Error_DNSServerNotResolved</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Error_Internal</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>Error_Other</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>Vendor-defined</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

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 bmsns1.xsd just for A_ARG_TYPE_NSLookupResult. In BasicManagement:2 this was incorporated into bms.xsd. For backwards compatibility (see section
2.5.9), bmsns1.xsd has to be used when a control point invokes a BasicManagement:1 action to return the result of a DNS lookup test.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<basicNsLookupResult xmlns:bms="urn:schemas-upnp-org:dm:bms"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:schemas-upnp-org:dm:bms
                        http://www.upnp.org/schemas/dm/bms.xsd">
  <Result>
    <!-- Optional Result element. -->
    <Status>Success</Status>
    <AnswerType>Authoritative</AnswerType>
    <HostNameReturned>a.b.c.d</HostNameReturned>
    <IPAddresses>1.2.3.4,2.3.4.5</IPAddresses>
    <DNSServerIP>33.44.55.66</DNSServerIP>
    <ResponseTime>255</ResponseTime>
  </Result>
</basicNsLookupResult>
```

This is an example XML document:
<IPAddresses>1.2.3.4, 2.3.4.5</IPAddresses>
<DNSServerIP>33.44.55.66</DNSServerIP>
<ResponseTime>255</ResponseTime>
</Result>
</bms:NSLookupResult>

### 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: allowedValueList for A_ARG_TYPE_TracerouteStatus**

<table>
<thead>
<tr>
<th>Value</th>
<th>Req. or Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Success</strong></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td><strong>Error_CannotResolveHostName</strong></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td><strong>Error_MaxHopCountExceeded</strong></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td><strong>Error_Internal</strong></td>
<td>O</td>
<td></td>
</tr>
<tr>
<td><strong>Error_Other</strong></td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>Vendor-defined</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

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.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestInfo xmlns:bms="urn:schemas-upnp-org:dm:bms"/>
```

Copyright UPnP Forum © 2012. All rights reserved.
<!-- The document contains one or more Protocol elements, each of which describes a different bandwidth test protocol. (Name, Version, Profile) is a unique key. There MUST be a Protocol element for each supported (Name, Version, Profile). See section 2.6.2. -->

<Protocol Name="Required bandwidth test protocol name, e.g. HTTP, FTP, Iperf, Echo, EchoPlus."
Version="Optional protocol version, e.g. '1.1'. Defaults to an empty string."
Profile="Required profile name, e.g. Baseline or BBF. Note that a mandatory Baseline profile is defined for each test protocol that is defined in this document. See section 2.6.6."
BaseProfiles="Optional base profiles names. XML (space-separated) list of the profiles that are extended by Profile. Defaults to an empty list, which therefore indicates that Profile doesn’t extend any other profiles."->

<Server>

<!-- Server-side information for this protocol (can be omitted if there is no server-side support for this protocol). Client-side information has exactly the same format (see below).

There are three types of capabilities: "Settings", "Files" and "Results". These are followed by any server instances. See section 2.6.3. -->

<Server>

<!-- Settings: parameters that can in theory be set for each test instance. All settings are optional as far as the XML Schema is concerned but might be required by some profiles. For each setting, the supported range or values can optionally be specified. See sections 2.6.4 and 2.6.6. -->

<!-- IP interface name. -->

<Interface Settable="Optional indication of whether or not the control point can set this parameter. If this attribute is omitted, the control point can set the parameter if there is more than one supported value."->

<AllowedValueList>

<AllowedValue>Allowed parameter value.</AllowedValue>
<AllowedValue>Zero or more additional AllowedValue elements.</AllowedValue>
</AllowedValueList>

</Interface>

<!-- All other settings follow the same pattern. For numeric parameters, both an allowed value list and an allowed value
range are permitted. For string and Boolean parameters, only an allowed value list is permitted. -->

<Transport/> <!-- Transport protocol name. -->
<Host/> <!-- Host name or IP address. -->
<Port/> <!-- Non-TLS listening port. -->
<TLSPort/> <!-- TLS listening port. -->
<SockBuffBytes/> <!-- Socket buffer size (bytes). -->
<AppBuffBytes/> <!-- Application buffer size (bytes). -->
<Direction/> <!-- Test data flow direction from the client point of view. -->
<TimeMSecs/> <!-- Test time (milliseconds). -->
<LengthBytes/> <!-- Test length (bytes). -->
<Username/> <!-- Usernames. -->
>Password/> <!-- Passwords. -->
<Path/> <!-- Paths. -->
<BandwidthBytesPerSec/> <!-- Desired bandwidth (bytes per sec). -->
<DSCP/> <!-- Diffserv code point. -->
<EthernetPriority/> <!-- Ethernet priority. -->
<IntervalMSecs/> <!-- Reporting interval (milliseconds). -->
<Threads/> <!-- Number of parallel threads. -->
<TCPNoDelay/> <!-- Nagle's algorithm control. -->
<TCPMSS/> <!-- Maximum segment size. -->
<ExitWhenDone/> <!-- Server/client exit control. -->

<!-- Files: optional details of data sources and sinks. See section 2.6.3. -->
<File>
  <Path>Required path formatted using "/" delimiters as specified in RFC 3986. This path is relative to the directory that the server has chosen to use for bandwidth test data.</Path>
  <RealFile>Optional indication of whether Path identifies a real file. Defaults to 1 (real file).</RealFile>
  <SizeBytes>Optional file size in bytes. Only applies to data sources that are real files.</SizeBytes>
  <MIMEType>Optional MIME type. Only applies to data sources.</MIMEType>
</File>

<File>Zero or more additional File elements.</File>

<!-- Results: required list of supported result parameters. See section 2.6.5. -->
<TCPSYNTime/> <!-- Time of initial TCP SYN. -->
<TCPSYNACKTime/> <!-- Time of initial TCP SYN ACK. -->
<TCPLateACKs/> <!-- Number of late TCP ACKs. -->
<ROCTime/> <!-- Receipt Of Message time. -->
<BOMTime/> <!-- Beginning Of Message time. -->
<EOMTime/> <!-- End Of Message time. -->
<TestBytesSent/> <!-- Test bytes sent. -->
<TestBytesReceived/> <!-- Test bytes received. -->
>TotalBytesSent/> <!-- Total bytes sent. -->
>TotalBytesReceived/> <!-- Total bytes received. -->
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.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestInfo xmlns:bms="urn:schemas-upnp-org:dm:bms"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <Protocol Name="Iperf" Version="2.0.5" Profile="Baseline">
    <Server>
      <!-- Both TCP and UDP are supported. Because more than value is supported, the control point can choose the transport. -->
      <Transport>
        <AllowedValueList>
          <AllowedValue>TCP</AllowedValue>
          <AllowedValue>UDP</AllowedValue>
        </AllowedValueList>
      </Transport>
      <!-- The host to which the server will bind. No list of allowed values is specified, so the control point can choose the value. -->
      <Host/>
    </Server>
  </Protocol>
</bms:BandwidthTestInfo>
```
<!-- Only port 5001 is supported. Therefore the control point cannot choose the port (it could specify 5001 but there is no need to do this). -->

<Port>
  <AllowedValueList>
    <AllowedValue>5001</AllowedValue>
  </AllowedValueList>
</Port>

<!-- These are the other supported settings. None specify allowed values or ranges, so the control point can choose the values for all of them. -->

<SockBuffBytes/>
<AppBuffBytes/>
<IntervalMSecs/>
<Threads/>
<TCPNoDelay/>
<TCPMSS/>
<ExitWhenDone/>

<!-- These are supported result parameters. -->

<TestBytesSent/>
<TestBytesReceived/>
<TestPacketsSent/>
<TestPacketsReceived/>
<LostPackets/>
<JitterUSecs/>

<!-- This is the single active Iperf server instance. -->

<Instance>
  <Transport>TCP</Transport>
  <Host>192.168.1.1</Host>
  <Port>5001</Port>
  <SockBuffBytes>65536</SockBuffBytes>
  <AppBuffBytes>131072</AppBuffBytes>
  <IntervalMSecs>1000</IntervalMSecs>
  <Threads>2</Threads>
  <TCPNoDelay>0</TCPNoDelay>
  <TCPMSS>1460</TCPMSS>
  <ExitWhenDone>0</ExitWhenDone>
</Instance>
</Server>
</Protocol>
</bms:BandwidthTestInfo>

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestInfo xmlns:bms="urn:schemas-upnp-org:dm:bms"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:schemas-upnp-org:dm:bms
    http://www.upnp.org/schemas/dm/bms.xsd">
  <Protocol Name="HTTP" Version="1.1" Profile="BBF"
    BaseProfiles="Baseline">
    <Client>
      <Transport>
```
<AllowedValueList>
  <AllowedValue>TCP</AllowedValue>
</AllowedValueList>
</Transport>
<Host/>
<Port/>
<Direction>
  <AllowedValueList>
    <AllowedValue>Get</AllowedValue>
  </AllowedValueList>
</Direction>
<Path/>
<DSCP/>
<EthernetPriority/>
<ROMTime/>
<BOMTime/>
<TestBytesReceived/>
>TotalBytesReceived/>
</Client>
</Protocol>
</bms:BandwidthTestInfo>

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.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestSpec xmlns:bms="urn:schemas-upnp-org:dm:bms"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

<!-- The document contains a single Protocol element which specifies a bandwidth test. The same bandwidth test specification is sent to both the server and the client, so any server-specific or client-specific settings are within Server or Client elements. -->
```
<Protocol Name="Required bandwidth test protocol name, e.g. HTTP, FTP, iPerf, Echo, EchoPlus. See section 2.6.2."
Version="Optional protocol version prefix, e.g. 1.1. If it matches more than one supported version, the latest matching version MUST be assumed. Defaults to an empty string, which therefore indicates the latest version."
Profile="Required profile name, e.g. Baseline or BBF. Requests that the test be performed in compliance with the requirements of the specified profile. See section 2.6.6."
BaseProfiles="Optional base profile names. If present, is an XML list (space-separated) specifying the names of the profiles that are extended by Profile."">

</Protocol>

<Interface>IP interface name.</Interface>
<Transport>Transport protocol name.</Transport>
<Host>Host name or IP address.</Host>
<Port>Non-TLS listening port.</Port>
<TLSPort>TLS listening port.</TLSPort>
<SockBuffBytes>Socket buffer size (bytes).</SockBuffBytes>
<AppBuffBytes>Application buffer size (bytes).</AppBuffBytes>
<Direction>Test data flow direction from the client point of view.</Direction>
<TimeMSecs>Test time (milliseconds).</TimeMSecs>
<LengthBytes>Test length (bytes).</LengthBytes>
<Username>Username.</Username>
<Password>Password.</Password>
<Path>Path.</Path>
</Server>
<!-- Server-specific settings are exactly the same as those at the outer level. If a setting is specified at both the outer level and within a <Server> element, the outer setting applies to the client (if relevant to the client) and the <Server> setting applies to the server. -->
</Server>

<!-- Client-specific settings behave exactly the same as server-specific settings. -->
</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.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestSpec xmlns:bms="urn:schemas-upnp-org:dm:bms"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:schemas-upnp-org:dm:bms
http://www.upnp.org/schemas/dm/bms.xsd">
    <Protocol Name="HTTP" Version="1.1" Profile="BBF">
        <Host>192.168.1.1</Host>
        <Port>80</Port>
        <Direction>Get</Direction>
        <Path>data/test.dat</Path>
    </Protocol>
</bms:BandwidthTestSpec>
```

The following more complicated example XML document describes a bidirectional **Iperf** test. It is equivalent to the following command lines:

- **Server**: `iperf -p 5001 -i 2 -s`
- **Client**: `iperf -p 5001 -i 2 -c 192.168.1.1 -L 5002 -d -t 10 -F data/test.dat`

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestSpec xmlns:bms="urn:schemas-upnp-org:dm:bms"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:schemas-upnp-org:dm:bms
http://www.upnp.org/schemas/dm/bms.xsd">
    <Protocol Name="Iperf" Version="2" Profile="Baseline">
        <Transport>TCP</Transport>
        <Port>5001</Port>
        <IntervalMSecs>2000</IntervalMSecs>
        <ExitWhenDone>1</ExitWhenDone>
        <Client>
            <Host>192.168.1.1</Host>
            <Port>5002</Port>
            <Direction>PutEcho</Direction>
            <TimeMSecs>10000</TimeMSecs>
            <Path>data/test.dat</Path>
            <Threads>2</Threads>
        </Client>
    </Protocol>
</bms:BandwidthTestSpec>
```
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: allowedValueList for A_ARG_TYPE_BandwidthTestStatus**

<table>
<thead>
<tr>
<th>Value</th>
<th>Req. or Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Success</strong></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Error_InitConnectionFailed</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Error_NoResponse</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Error_TransferFailed</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Error_PasswordRequestFailed</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Error_LoginFailed</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Error_NoTransferMode</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Error_IncorrectSize</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Error_NoTransferComplete</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Error_Timeout</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Error_Other</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Error_FTPNoPASV</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Error_FTPNoCWD</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Error_FTPNoSTOR</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Vendor-defined</td>
<td>X^1</td>
<td></td>
</tr>
</tbody>
</table>

^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.
<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestResult xmlns:bms="urn:schemas-upnp-org:dm:bms"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
<!-- The document contains a single Protocol element which describes a bandwidth test result. See section 2.6.2. -->
<Protocol Name="Required bandwidth test protocol name, e.g. HTTP, FTP, Iperf, Echo, EchoPlus. MUST be the value that was specified in the test specification."
Version="Optional protocol version, e.g. 1.1. MUST be the version that matched the value that was specified in the test specification."
Profile="Required profile name, e.g. Baseline or BBF. MUST be the value that was specified in the test specification. See section 2.6.6.">
<!-- The test specification -->
<Spec>
<!-- The contents of the test specification’s <Protocol> element plus anything that was defaulted (so this is the complete test specification). -->
</Spec>
<!-- Server-side and client-side results have exactly the same format and can be interleaved as desired, e.g. a set of client-side results can be followed by a set of server-side results and then another set of client-side results. -->
<Server Interval="Optional indication of the interval for these results. Only affects tests with multiple reporting intervals. Allows separate reporting per interval. Has the syntax 'start:end', where start and end are elapsed milliseconds from the start of the test. If omitted, the results are summed across all intervals."
Stream="Optional indication of which stream these results are associated with. This is an opaque identifier whose interpretation is a function of the protocol and/or the implementation. For example it might be a socket number or a thread identifier. If omitted, the results are summed across all streams.">
<!-- Individual results can be omitted if they are not relevant to this test. Profile definitions can provide more specific requirements. See sections 2.6.5 and 2.6.6. -->
<TCPSYNTIme>Time of initial TCP SYN.</TCPSYNTIme>
<TCPSYNACKTime>Time of initial TCP SYN ACK.</TCPSYNACKTime>
<TCPLateACKs>Number of late TCP ACKs.</TCPLateACKs>
The following example document shows the result of an HTTP/1.1 Get (download) test.

```xml
<xml version="1.0" encoding="UTF-8">
<bms:BandwidthTestResult xmlns:bms="urn:schemas-upnp-org:dm:bms"
xmns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <Protocol Name="HTTP" Version="1.1" Profile="Baseline">
        <Spec>
            <Transport>TCP</Transport>
            <Host>192.168.1.1</Host>
            <Port>80</Port>
            <Direction>Get</Direction>
            <Path>data/test.dat</Path>
        </Spec>
        <Client>
            <TotalBytesReceived>123456</TotalBytesReceived>
        </Client>
    </Protocol>
</bms:BandwidthTestResult>
```

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

```xml
<xml version="1.0" encoding="UTF-8">
<bms:BandwidthTestResult xmlns:bms="urn:schemas-upnp-org:dm:bms"
xmns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <Protocol Name="Iperf" Profile="Baseline">
        <Spec>
            <Transport>TCP</Transport>
            <Host>192.168.1.1</Host>
            <Port>80</Port>
            <Direction>Get</Direction>
            <Path>data/test.dat</Path>
        </Spec>
        <Client>
            <TotalBytesReceived>123456</TotalBytesReceived>
        </Client>
    </Protocol>
</bms:BandwidthTestResult>
```
<Spec>
  <Transport>TCP</Transport>
  <Host>192.168.1.1</Host>
  <Port>5001</Port>
  <SockBuffBytes>2048</SockBuffBytes>
  <AppBuffBytes>2048</AppBuffBytes>
  <Direction>Put</Direction>
  <TimeMSecs>10000</TimeMSecs>
  <IntervalMSecs>2000</IntervalMSecs>
  <Threads>1</Threads>
  <TCPNoDelay>0</TCPNoDelay>
  <TCPMSS>1460</TCPMSS>
  <ExitWhenDone>0</ExitWhenDone>
</Spec>

<Client Interval="0:2000">
  <TestBytesSent>123456</TestBytesSent>
</Client>

<Client Interval="2000:4000">
  <TestBytesSent>123456</TestBytesSent>
</Client>

<Client Interval="4000:6000">
  <TestBytesSent>123456</TestBytesSent>
</Client>

<Client Interval="6000:8000">
  <TestBytesSent>123456</TestBytesSent>
</Client>

<Client Interval="8000:10000">
  <TestBytesSent>123456</TestBytesSent>
</Client>

<Client>
  <TestBytesSent>617280</TestBytesSent>
</Client>

</Protocol>
</bms:BandwidthTestResult>

### 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

<table>
<thead>
<tr>
<th>Value</th>
<th>Req. or Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllInterfaces</td>
<td>R</td>
<td>All IP interfaces</td>
</tr>
<tr>
<td>RequestInterface</td>
<td>R</td>
<td>The IP interface on which the action request was received is to be reset.</td>
</tr>
<tr>
<td>NorthboundInterfaces</td>
<td>O</td>
<td>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.</td>
</tr>
<tr>
<td>Value</td>
<td>Req. or Opt.</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Vendor-defined</td>
<td>X¹</td>
<td>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 /UPnP/DM/Configuration/Network/IPInterface/#/SystemName parameter.</td>
</tr>
</tbody>
</table>

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.

#### Table 2-11: allowedValueList for **A_ARG_TYPE_InterfaceResetStatus**

<table>
<thead>
<tr>
<th>Value</th>
<th>Req. or Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Error Other</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Vendor-defined</td>
<td>X¹</td>
<td></td>
</tr>
</tbody>
</table>

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.

**Table 2-12: allowedValueList for A_ARG_TYPE_LogLevel**

<table>
<thead>
<tr>
<th>Value</th>
<th>Req. or Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Emergency</em></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td><em>Alert</em></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td><em>Critical</em></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td><em>Error</em></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td><em>Warning</em></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td><em>Notice</em></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td><em>Informational</em></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td><em>Debug</em></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td><em>Vendor-defined</em></td>
<td>X^2</td>
<td></td>
</tr>
</tbody>
</table>

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.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<XMLNS xmlns:bms="urn:schemas-upnp-org:dm:bms"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:schemas-upnp-org:dm:bms"
http://www.upnp.org/schemas/dm/bms.xsd">
```
<!-- Contains zero or more ACLEntry elements, each of which has a unique (Action, Filter). -->

<ACLEntry> <!-- Optional ACLEntry element. -->

<Action>Required name of the action to which this ACL entry refers.</Action>

<Filter>Required ACL filter expression that determines whether this entry matches a given action invocation. Filter syntax is defined below.</Filter>

<Roles>Required space-separated list of Roles that are permitted to invoke this action if the filter matches.</Roles>

</ACLEntry>

<ACLEntry>Zero or more additional ACLEntry elements.</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 “:restricted:”.

<?xml version="1.0" encoding="UTF-8"?>
<bms:ACL xmlns:bms="urn:schemas-upnp-org:dm:bms"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <ACLEntry>
    <Action>InterfaceReset</Action>
    <Filter>Interfaces="RequestInterface"</Filter>
    <Roles>Basic</Roles>
  </ACLEntry>
  <ACLEntry>
    <Action>SetLogInfo</Action>
    <Filter>contains(LogURI,":restricted:"")</Filter>
    <Roles>Basic</Roles>
  </ACLEntry>
</bms:ACL>

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)

Copyright UPnP Forum © 2012. All rights reserved.
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:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bms:ACL xmlns:bms="urn:schemas-upnp-org:dm:bms"
xmlns:ex="http://www.example.com/upnp-dm-bms" ...

...<ACLEntry>
   <Action>X_Example_VendorTest</Action>
   <Filter>ex:urlSchemeIs(Host,"https") and DSCP=0</Filter>
   <Roles>Public</Roles>
</ACLEntry>
</bms: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 `ex:urlSchemeIs()` function did.

### 2.4. Eventing and Moderation

**Table 2-13: Event Moderation**

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Evented</th>
<th>Moderated Event</th>
<th>Max Event Rate¹</th>
<th>Logical Combination</th>
<th>Min Delta per Event²</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DeviceStatus</code></td>
<td>Yes</td>
<td>Yes</td>
<td>1.0 second</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>SequenceMode</code></td>
<td>Yes</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>TestIDs</code></td>
<td>Yes</td>
<td>Yes</td>
<td>0.2 seconds</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ActiveTestIDs</code></td>
<td>Yes</td>
<td>Yes</td>
<td>0.2 seconds</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>LogURIs</code></td>
<td>Yes</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Non-standard state variables implemented by an UPnP vendor go here.</code></td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
</tr>
</tbody>
</table>

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

---

Copyright UPnP Forum © 2012. All rights reserved.
### 2.4.1. SSDP Announcement Mechanism

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**

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Req. or Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AboutToReboot</td>
<td>R</td>
<td>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 “1” for various reasons, including autonomous reboots and Reboot() requests.</td>
</tr>
<tr>
<td>AboutToBaselineReset</td>
<td>R</td>
<td>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 “1” for various reasons, including autonomous baseline resets and BaselineReset() requests.</td>
</tr>
<tr>
<td>Vendor-defined</td>
<td>X</td>
<td>Any vendor-defined values MUST obey the usual naming rules for vendor extensions, as defined in [UDA1.0].</td>
</tr>
</tbody>
</table>

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 “1” 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 “1” when ssdp:alive messages are sent, so Announcement.dm.upnp.org: AboutToReboot and Announcement.dm.upnp.org: AboutToBaselineReset headers are likely to be present only in ssdp:byebye messages. However, the use of the Announcement.dm.upnp.org mechanism with ssdp:alive messages is not specifically forbidden.

Copyright UPnP Forum © 2012. All rights reserved.
2.5. Actions

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.

Table 2-15: Actions

<table>
<thead>
<tr>
<th>Name</th>
<th>Device R/O</th>
<th>Control Point R/O</th>
<th>Recommended Role List</th>
<th>Recommended Restricted Role List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reboot()</td>
<td>O</td>
<td>O</td>
<td>Admin</td>
<td></td>
</tr>
<tr>
<td>BaselineReset()</td>
<td>O</td>
<td>O</td>
<td>Admin</td>
<td></td>
</tr>
<tr>
<td>GetDeviceStatus()</td>
<td>R</td>
<td>O</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>GetSequenceMode()</td>
<td>O</td>
<td>O</td>
<td>Admin, Basic</td>
<td>Public</td>
</tr>
<tr>
<td>GetSequenceMode()</td>
<td>O</td>
<td>O</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>Ping()</td>
<td>CR 5</td>
<td>CR 5</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>GetPingResult()</td>
<td>CR 5</td>
<td>CR 5</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>NSLookup()</td>
<td>CR 6</td>
<td>CR 6</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>GetNSLookupResult()</td>
<td>CR 6</td>
<td>CR 6</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>Traceroute()</td>
<td>CR 7</td>
<td>CR 7</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>GetTracerouteResult()</td>
<td>CR 7</td>
<td>CR 7</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>GetBandwidthTestInfo()</td>
<td>CR 8</td>
<td>CR 8</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>BandwidthTest()</td>
<td>CR 8</td>
<td>CR 8</td>
<td>Admin, Basic</td>
<td></td>
</tr>
<tr>
<td>GetBandwidthTestResult()</td>
<td>CR 8</td>
<td>CR 8</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>InterfaceReset()</td>
<td>CR 9</td>
<td>CR 9</td>
<td>Admin, Basic</td>
<td></td>
</tr>
<tr>
<td>GetInterfaceResetResult()</td>
<td>CR 9</td>
<td>CR 9</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>SelfTest()</td>
<td>CR 10</td>
<td>CR 10</td>
<td>Admin, Basic</td>
<td></td>
</tr>
<tr>
<td>GetSelfTestResult()</td>
<td>CR 10</td>
<td>CR 10</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>GetTestIDs()</td>
<td>CR 11</td>
<td>CR 11</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>GetActiveTestIDs()</td>
<td>CR 11</td>
<td>CR 11</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Device R/O</td>
<td>Control Point R/O</td>
<td>Recommended Role List</td>
<td>Recommended Restricted Role List</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------</td>
<td>-------------------</td>
<td>-----------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>GetTestInfo()</td>
<td>CR</td>
<td>CR</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>CancelTest()</td>
<td>CR</td>
<td>CR</td>
<td>Admin</td>
<td>Basic</td>
</tr>
<tr>
<td>GetLogURIs()</td>
<td>O</td>
<td>O</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>SetLogInfo()</td>
<td>O</td>
<td>O</td>
<td>Admin</td>
<td>Basic</td>
</tr>
<tr>
<td>GetLogInfo()</td>
<td>O</td>
<td>O</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>GetACLData()</td>
<td>CR</td>
<td>CR</td>
<td>Admin</td>
<td>Basic</td>
</tr>
<tr>
<td><strong>Non-standard actions implemented by an UPnP vendor go here.</strong></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

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 ssdp:byebye 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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>RebootStatus</td>
<td>OUT</td>
<td>A_ARG_TYPE_RebootStatus</td>
</tr>
</tbody>
</table>

### 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 /UPnP/DM/DeviceInfo/ExecutionEnvironment/WillReboot [CMS] parameter is present and has the value “1”, the Reboot() action MUST reboot the targeted Execution Environment. If [CMS] is not supported, the parameter is absent, or it has the value “0”, control points cannot determine whether or not the Reboot() action will reboot the targeted Execution Environment.

- If the /UPnP/DM/DeviceInfo/OperatingSystem/WillReboot [CMS] parameter is present and has the value “1”, the Reboot() action MUST reboot the Operating System. If [CMS] is not supported, the parameter is absent, or it has the value “0”, 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 “1” (section 2.4.1), meaning that each ssdp:byebye message will include an Announcement.dm.upnp.org: - AboutToReboot header.

Once the Parent Device AboutToReboot internal state has been set to “1”:

- 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.

Copyright UPnP Forum © 2012. All rights reserved.
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

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

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>606</td>
<td>Action not authorized</td>
<td>The action requested requires authorization and the sender was not authorized.</td>
</tr>
</tbody>
</table>

### 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 `ssdp:byebye` 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 `/UPnP/DM/DeviceInfo/ExecutionEnvironment/WillBaselineReset` [CMS] parameter is present and has the value “1”, 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 “0”, control points cannot determine whether or not the *BaselineReset()* action will return the targeted Execution Environment to its baseline state.

- If the /UPnP/DM/DeviceInfo/OperatingSystem/WillBaselineReset [CMS] parameter is present and has the value “1”, 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 “0”, 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 “1” (section 2.4.1), meaning that each ssdp:byebye 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

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

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>606</td>
<td>Action not authorized</td>
<td>The action requested requires authorization and the sender was not authorized.</td>
</tr>
</tbody>
</table>

### 2.5.3. *GetDeviceStatus()*

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

#### 2.5.3.1. Arguments

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeviceStatus</td>
<td>OUT</td>
<td>DeviceStatus</td>
</tr>
</tbody>
</table>

#### 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()

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
</tbody>
</table>

2.5.4.  SetSequenceMode()

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

- **NewSequenceMode**: is the new value of the SequenceMode state variable.

- **OldSequenceMode**: returns the previous value of the SequenceMode state variable, so a control point that sets SequenceMode to “1” will know whether another control point had already set it to “1”.

2.5.4.1.  Arguments

Table 2-21: Arguments for SetSequenceMode()

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewSequenceMode</td>
<td>IN</td>
<td>SequenceMode</td>
</tr>
<tr>
<td>OldSequenceMode</td>
<td>OUT</td>
<td>SequenceMode</td>
</tr>
</tbody>
</table>

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()

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
</tbody>
</table>
2.5.5. GetSequenceMode()

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

2.5.5.1. Arguments

Table 2-23: Arguments for GetSequenceMode()

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SequenceMode</td>
<td>OUT</td>
<td>SequenceMode</td>
</tr>
</tbody>
</table>

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

Table 2-24: Error Codes for GetSequenceMode()

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>606</td>
<td>Action not authorized</td>
<td>The action requested requires authorization and the sender was not authorized.</td>
</tr>
</tbody>
</table>

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:

- **Host**: is the name or address of the ICMP echo packet destination. It MUST NOT be an empty string.
- **NumberOfRepetitions**: is the number of packets to send. A value of zero requests use of an implementation-chosen default number of repetitions.
- **Timeout**: 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.
• **DataBlockSize**: 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.

• **DSCP**: 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

Table 2-25: Arguments for Ping()

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>IN</td>
<td>A_ARG_TYPE_Host</td>
</tr>
<tr>
<td>NumberOfRepetitions</td>
<td>IN</td>
<td>A_ARG_TYPE_UInt</td>
</tr>
<tr>
<td>Timeout</td>
<td>IN</td>
<td>A_ARG_TYPE_MSecs</td>
</tr>
<tr>
<td>DataBlockSize</td>
<td>IN</td>
<td>A_ARG_TYPE_USHORT</td>
</tr>
<tr>
<td>DSCP</td>
<td>IN</td>
<td>A_ARG_TYPE_DSCP</td>
</tr>
<tr>
<td>TestID</td>
<td>OUT</td>
<td>A_ARG_TYPE_TestID</td>
</tr>
</tbody>
</table>

### 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 **Timeout** 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

Table 2-26: Error Codes for Ping()

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>703</td>
<td>Test Already Active</td>
<td>A test of this type is already active (and the implementation doesn’t support multiple active instances of a given test type).</td>
</tr>
<tr>
<td>704</td>
<td>Capabilities Preclude Test</td>
<td>Test arguments are individually valid but, taken together, describe a test that is beyond the service’s capabilities.</td>
</tr>
<tr>
<td>705</td>
<td>State Precludes Test</td>
<td>Service state precludes performing this test.</td>
</tr>
</tbody>
</table>
2.5.7. **GetPingResult()**

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

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

### Arguments

**Table 2-27: Arguments for GetPingResult()**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestID</td>
<td>IN</td>
<td>A_ARG_TYPE_TestID</td>
</tr>
<tr>
<td>Status</td>
<td>OUT</td>
<td>A_ARG_TYPE_PingStatus</td>
</tr>
<tr>
<td>AdditionalInfo</td>
<td>OUT</td>
<td>A_ARG_TYPE_String</td>
</tr>
<tr>
<td>SuccessCount</td>
<td>OUT</td>
<td>A_ARG_TYPE_UInt</td>
</tr>
<tr>
<td>FailureCount</td>
<td>OUT</td>
<td>A_ARG_TYPE_UInt</td>
</tr>
<tr>
<td>AverageResponseTime</td>
<td>OUT</td>
<td>A_ARG_TYPE_MSecs</td>
</tr>
<tr>
<td>MinimumResponseTime</td>
<td>OUT</td>
<td>A_ARG_TYPE_MSecs</td>
</tr>
<tr>
<td>MaximumResponseTime</td>
<td>OUT</td>
<td>A_ARG_TYPE_MSecs</td>
</tr>
</tbody>
</table>

### Device Requirements

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

### Dependency on State

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

### Effect on State

None.

### Errors

**Table 2-28: Error Codes for GetPingResult()**
<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>706</td>
<td>No Such Test</td>
<td>No test with the specified TestID was found.</td>
</tr>
<tr>
<td>707</td>
<td>Wrong Test Type</td>
<td>TestID is valid but refers to a different test type.</td>
</tr>
<tr>
<td>708</td>
<td>Invalid Test State</td>
<td>The TestID is valid but test results are not available.</td>
</tr>
</tbody>
</table>

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:

- **HostName**: is the name of the host to look up. The current domain name MUST be used unless the name is a fully qualified name.
- **DNSServer**: 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.
- **NumberOfRepetitions**: 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.
- **Timeout**: 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**

Table 2-29: Arguments for **NSLookup()**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>HostName</td>
<td>IN</td>
<td>A_ARG_TYPE_HostName</td>
</tr>
<tr>
<td>DNSServer</td>
<td>IN</td>
<td>A_ARG_TYPE_Host</td>
</tr>
<tr>
<td>NumberOfRepetitions</td>
<td>IN</td>
<td>A_ARGTypeEnum_UInt</td>
</tr>
<tr>
<td>Timeout</td>
<td>IN</td>
<td>A_ARG_TYPE_MSecs</td>
</tr>
<tr>
<td>TestID</td>
<td>OUT</td>
<td>A_ARG_TYPE_TestID</td>
</tr>
</tbody>
</table>

**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**

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

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:

- **Status**: indicates the overall success or failure of the test (if the test failed, the values of the remaining output arguments – other than **AdditionalInfo** – are not specified, and MUST be ignored).
- **AdditionalInfo**: a free-format string that can contain additional information about the test result.
- **SuccessCount**: is the number of successful DNS lookups (those for which a successful response was received prior to the timeout).
- **Result**: 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: Arguments for **GetNSLookupResult**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestID</td>
<td><strong>IN</strong></td>
<td><strong>A_ARG_TYPE_TestID</strong></td>
</tr>
<tr>
<td>Status</td>
<td><strong>OUT</strong></td>
<td><strong>A_ARG_TYPE_NSLookup-Status</strong></td>
</tr>
<tr>
<td>AdditionalInfo</td>
<td><strong>OUT</strong></td>
<td><strong>A_ARG_TYPE_String</strong></td>
</tr>
<tr>
<td>SuccessCount</td>
<td><strong>OUT</strong></td>
<td><strong>A_ARG_TYPE_UInt</strong></td>
</tr>
<tr>
<td>Result</td>
<td><strong>OUT</strong></td>
<td><strong>A_ARG_TYPE_NSLookup-Result</strong></td>
</tr>
</tbody>
</table>

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.

Copyright UPnP Forum © 2012. All rights reserved.
2.5.9.4. **Effect on State**

None.

2.5.9.5. **Errors**

Table 2-32: Error Codes for `GetNSLookupResult()`

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>706</td>
<td>No Such Test</td>
<td>No test with the specified TestID was found.</td>
</tr>
<tr>
<td>707</td>
<td>Wrong Test Type</td>
<td>TestID is valid but refers to a different test type.</td>
</tr>
<tr>
<td>708</td>
<td>Invalid Test State</td>
<td>The TestID is valid but test results are not available.</td>
</tr>
</tbody>
</table>

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:

- **Host**: is the name or address of the host to find a route to. It MUST NOT be an empty string.
- **Timeout**: is the length of time (in milliseconds) to wait for each reply. A value of zero requests use of an implementation-chosen timeout.
- **DataBlockSize**: 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.
- **MaxHopCount**: 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.
- **DSCP**: 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()`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>IN</td>
<td><code>A_ARG_TYPE_Host</code></td>
</tr>
<tr>
<td>Timeout</td>
<td>IN</td>
<td><code>A_ARG_TYPE_MSecs</code></td>
</tr>
<tr>
<td>DataBlockSize</td>
<td>IN</td>
<td><code>A_ARG_TYPE_USHORT</code></td>
</tr>
<tr>
<td>MaxHopCount</td>
<td>IN</td>
<td><code>A_ARG_TYPE_UINT</code></td>
</tr>
<tr>
<td>DSCP</td>
<td>IN</td>
<td><code>A_ARG_TYPE_DSCP</code></td>
</tr>
<tr>
<td>TestID</td>
<td>OUT</td>
<td><code>A_ARG_TYPE_TestID</code></td>
</tr>
</tbody>
</table>
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

Table 2-34: Error Codes for Traceroute()

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

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:

- **Status**: indicates the overall success or failure of the test (if the test failed, the values of the remaining output arguments – other than AdditionalInfo – are not specified, and MUST be ignored).
- **AdditionalInfo**: a free-format string that can contain additional information about the test result.
- **ResponseTime**: is the average response time (in milliseconds) for the most recent probe, i.e. for the messages that actually reached the host.
- **HopHosts**: 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

Table 2-35: Arguments for GetTracerouteResult()

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestID</td>
<td>IN</td>
<td>A ARG TYPE TestID</td>
</tr>
<tr>
<td>Status</td>
<td>OUT</td>
<td>A ARG TYPE Traceroute-Status</td>
</tr>
<tr>
<td>AdditionalInfo</td>
<td>OUT</td>
<td>A ARG TYPE String</td>
</tr>
</tbody>
</table>
### 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

#### Table 2-36: Error Codes for \textit{GetTracerouteResult()}

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>706</td>
<td>No Such Test</td>
<td>No test with the specified TestID was found.</td>
</tr>
<tr>
<td>707</td>
<td>Wrong Test Type</td>
<td>TestID is valid but refers to a different test type.</td>
</tr>
<tr>
<td>708</td>
<td>Invalid Test State</td>
<td>The TestID is valid but test results are not available.</td>
</tr>
</tbody>
</table>

### 2.5.12. \textit{GetBandwidthTestInfo()}

The \textit{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

#### Table 2-37: Arguments for \textit{GetBandwidthTestInfo()}

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>BandwidthTestInfo</td>
<td>OUT</td>
<td>A ARG TYPE Bandwidth-TestInfo</td>
</tr>
</tbody>
</table>

### 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 \textit{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

Table 2-38: Error Codes for GetBandwidthTestInfo()

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
</tbody>
</table>

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:

- BandwidthTestSpec: 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).
- TestEndpoint: indicates whether the recipient should play the role of server or of client.
- TestSchedule: 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.
- TestSessID: 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

Table 2-39: Arguments for BandwidthTest()

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>BandwidthTestSpec</td>
<td>IN</td>
<td>A_ARG_TYPE_BandwidthTestSpec</td>
</tr>
<tr>
<td>TestEndpoint</td>
<td>IN</td>
<td>A_ARG_TYPE_TestEndpoint</td>
</tr>
<tr>
<td>TestSchedule</td>
<td>IN</td>
<td>A_ARG_TYPE_TestSchedule</td>
</tr>
<tr>
<td>TestSessID</td>
<td>IN</td>
<td>A_ARG_TYPE_TestSessID</td>
</tr>
</tbody>
</table>
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

Table 2-40: Error Codes for BandwidthTest()

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>606</td>
<td>Action not authorized</td>
<td>The action requested requires authorization and the sender was not authorized.</td>
</tr>
<tr>
<td>703</td>
<td>Test Already Active</td>
<td>A test of this type is already active (and the implementation doesn’t support multiple active instances of a given test type).</td>
</tr>
<tr>
<td>704</td>
<td>Capabilities Preclude Test</td>
<td>Test arguments are individually valid but, taken together, describe a test that is beyond the service’s capabilities.</td>
</tr>
<tr>
<td>705</td>
<td>State Precludes Test</td>
<td>Service state precludes performing this test.</td>
</tr>
<tr>
<td>712</td>
<td>Invalid Test Endpoint</td>
<td>The service does not support this test for the specified endpoint.</td>
</tr>
</tbody>
</table>

2.5.14. GetBandwidthTestResult()

The GetBandwidthTestResult() action returns the results of a completed bandwidth test or (if supported) the partial results of an active bandwidth test. The output arguments are defined as follows:

- **State**: indicates the current test state.
- **Status**: indicates the overall success or failure of the test.
- **AdditionalInfo**: a free-format string that can contain additional information about the test result.
• **Result:** 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

Table 2-41: Arguments for **GetBandwidthTestResult()**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestID</td>
<td>IN</td>
<td>A_ARG_TYPE_TestID</td>
</tr>
<tr>
<td>State</td>
<td>OUT</td>
<td>A_ARG_TYPE_TestState</td>
</tr>
<tr>
<td>Status</td>
<td>OUT</td>
<td>A_ARG_TYPE_Bandwidth-TestStatus</td>
</tr>
<tr>
<td>AdditionalInfo</td>
<td>OUT</td>
<td>A_ARG_TYPE_String</td>
</tr>
<tr>
<td>Result</td>
<td>OUT</td>
<td>A_ARG_TYPE_Bandwidth-TestResult</td>
</tr>
</tbody>
</table>

### 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

Table 2-42: Error Codes for **GetBandwidthTestResult()**

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>706</td>
<td>No Such Test</td>
<td>No test with the specified TestID was found.</td>
</tr>
<tr>
<td>707</td>
<td>Wrong Test Type</td>
<td>TestID is valid but refers to a different test type.</td>
</tr>
<tr>
<td>708</td>
<td>Invalid Test State</td>
<td>The TestID is valid but test results are not available.</td>
</tr>
</tbody>
</table>

### 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:

Copyright UPnP Forum © 2012. All rights reserved.
BasicManagement:2 Service Template Version 1.01

- **Interfaces**: 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

Table 2-43: Arguments for *InterfaceReset()*

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interfaces</td>
<td>IN</td>
<td>A_ARG_TYPE_Interfaces</td>
</tr>
<tr>
<td>TestID</td>
<td>OUT</td>
<td>A_ARG_TYPE_TestID</td>
</tr>
</tbody>
</table>

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

Table 2-44: Error Codes for *InterfaceReset()*

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
</tbody>
</table>
### 2.5.16. GetInterfaceResetResult()

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

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

**Table 2-45: Arguments for GetInterfaceResetResult()**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestID</td>
<td>IN</td>
<td><code>A_ARG_TYPE_TestID</code></td>
</tr>
<tr>
<td>Status</td>
<td>OUT</td>
<td><code>A_ARG_TYPE_InterfaceResetStatus</code></td>
</tr>
<tr>
<td>AdditionalInfo</td>
<td>OUT</td>
<td><code>A_ARG_TYPE_String</code></td>
</tr>
<tr>
<td>NumberOfSuccesses</td>
<td>OUT</td>
<td><code>A_ARG_TYPE_USHORT</code></td>
</tr>
<tr>
<td>NumberOfFailures</td>
<td>OUT</td>
<td><code>A_ARG_TYPE_USHORT</code></td>
</tr>
</tbody>
</table>

#### 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.

---

**errorCode**  **errorDescription**  **Description**

| 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 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.                                  |
2.5.16.4. Effect on State

None.

2.5.16.5. Errors

Table 2-46: Error Codes for `GetInterfaceResetResult()`

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>706</td>
<td>No Such Test</td>
<td>No test with the specified TestID was found.</td>
</tr>
<tr>
<td>707</td>
<td>Wrong Test Type</td>
<td>TestID is valid but refers to a different test type.</td>
</tr>
<tr>
<td>708</td>
<td>Invalid Test State</td>
<td>The TestID is valid but test results are not available.</td>
</tr>
</tbody>
</table>

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()`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestID</td>
<td>OUT</td>
<td><code>A_ARG_TYPE_TestID</code></td>
</tr>
</tbody>
</table>

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()`
### 2.5.18. GetSelfTestResult()

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

- **Status**: indicates whether the test succeeded (1) or failed (0).
- **AdditionalInfo**: a free-format string that can contain additional information about the test result.

#### 2.5.18.1. Arguments

Table 2-49: Arguments for **GetSelfTestResult()**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestID</td>
<td><strong>IN</strong></td>
<td>A_ARG_TYPE_TestID</td>
</tr>
<tr>
<td>Status</td>
<td><strong>OUT</strong></td>
<td>A_ARG_TYPE_Boolean</td>
</tr>
<tr>
<td>AdditionalInfo</td>
<td><strong>OUT</strong></td>
<td>A_ARG_TYPE_String</td>
</tr>
</tbody>
</table>

#### 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

Table 2-50: Error Codes for **GetSelfTestResult()**

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
</tbody>
</table>
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**

Table 2-51: Arguments for *GetTestIDs()*

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestIDs</td>
<td>OUT</td>
<td>TestIDs</td>
</tr>
</tbody>
</table>

**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**

Table 2-52: Error Codes for *GetTestIDs()*

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
</tbody>
</table>

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**

Table 2-53: Arguments for *GetActiveTestIDs()*

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestIDs</td>
<td>OUT</td>
<td>ActiveTestIDs</td>
</tr>
</tbody>
</table>
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
Table 2-54: Error Codes for GetActiveTestIIDs()

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
</tbody>
</table>

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

2.5.21.1. Arguments
Table 2-55: Arguments for GetTestInfo()

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestID</td>
<td>IN</td>
<td>A_ARG_TYPE_TestID</td>
</tr>
<tr>
<td>Type</td>
<td>OUT</td>
<td>A_ARG_TYPE_TestType</td>
</tr>
<tr>
<td>State</td>
<td>OUT</td>
<td>A_ARG_TYPE_TestState</td>
</tr>
</tbody>
</table>

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
Table 2-56: Error Codes for GetTestInfo()

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
</tbody>
</table>
2.5.22. **CancelTest()**

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

2.5.22.1. **Arguments**

Table 2-57: Arguments for `CancelTest()`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestID</td>
<td>IN</td>
<td>A_ARG_TYPE_TestID</td>
</tr>
</tbody>
</table>

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**

Table 2-58: Error Codes for `CancelTest()`

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>606</td>
<td>Action not authorized</td>
<td>The action requested requires authorization and the sender was not authorized.</td>
</tr>
<tr>
<td>706</td>
<td>No Such Test</td>
<td>No test with the specified TestID was found.</td>
</tr>
<tr>
<td>709</td>
<td>State Precludes Cancel</td>
<td>The TestID is valid but the test can’t be canceled.</td>
</tr>
</tbody>
</table>
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

Table 2-59: Arguments for **GetLogURIs()**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogURIs</td>
<td>OUT</td>
<td>LogURIs</td>
</tr>
</tbody>
</table>

### 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

Table 2-60: Error Codes for **GetLogURIs()**

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
</tbody>
</table>

2.5.24. **SetLogInfo()**

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

### 2.5.24.1. Arguments

Table 2-61: Arguments for **SetLogInfo()**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogURI</td>
<td>IN</td>
<td>A_ARG_TYPE_LogURI</td>
</tr>
<tr>
<td>Enabled</td>
<td>IN</td>
<td>A_ARG_TYPE_Boolean</td>
</tr>
<tr>
<td>LogLevel</td>
<td>IN</td>
<td>A_ARG_TYPE_LogLevel</td>
</tr>
</tbody>
</table>

### 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”.

Copyright UPnP Forum © 2012. All rights reserved.
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()

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>606</td>
<td>Action not authorized</td>
<td>The action requested requires authorization and the sender was not authorized.</td>
</tr>
<tr>
<td>710</td>
<td>No Such Log</td>
<td>No log with the specified LogURI was found.</td>
</tr>
<tr>
<td>711</td>
<td>Log Not Configurable</td>
<td>Log doesn’t permit enable/disable and/or log level to be changed.</td>
</tr>
</tbody>
</table>

### 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()

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogURI</td>
<td>IN</td>
<td>A_ARG_TYPE_LogURI</td>
</tr>
<tr>
<td>Configurable</td>
<td>OUT</td>
<td>A_ARG_TYPE_Boolean</td>
</tr>
<tr>
<td>Enabled</td>
<td>OUT</td>
<td>A_ARG_TYPE_Boolean</td>
</tr>
<tr>
<td>LogLevel</td>
<td>OUT</td>
<td>A_ARG_TYPE_LogLevel</td>
</tr>
<tr>
<td>LogURL</td>
<td>OUT</td>
<td>A_ARG_TYPE_LogURL</td>
</tr>
</tbody>
</table>
### Argument List for GetLogInfo()

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxSize</td>
<td>OUT</td>
<td>A_ARG_TYPE_LogMaxSize</td>
</tr>
<tr>
<td>LastChange</td>
<td>OUT</td>
<td>A_ARG_TYPE_DateTime</td>
</tr>
</tbody>
</table>

#### 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

**Table 2-64: Error Codes for GetLogInfo()**

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>710</td>
<td>No Such Log</td>
<td>No log with the specified LogURI was found.</td>
</tr>
</tbody>
</table>

#### 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

**Table 2-65: Arguments for GetACLData()**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL</td>
<td>OUT</td>
<td>A_ARG_TYPE_ACL</td>
</tr>
</tbody>
</table>

#### 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

Table 2-66: Error Codes for *GetACLData()*

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>606</td>
<td>Action not authorized</td>
<td>The action requested requires authorization and the sender was not authorized.</td>
</tr>
</tbody>
</table>

2.5.27. Common Error Codes
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.

Table 2-67: Common Error Codes

<table>
<thead>
<tr>
<th>errorCode</th>
<th>errorDescription</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-499</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>500-599</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>600-699</td>
<td>TBD</td>
<td>See UPnP Device Architecture section on Control.</td>
</tr>
<tr>
<td>606</td>
<td>Action not authorized</td>
<td>The action requested requires authorization and the sender was not authorized.</td>
</tr>
<tr>
<td>700</td>
<td></td>
<td>Reserved for future extensions.</td>
</tr>
<tr>
<td>701</td>
<td>Interface Not Found</td>
<td>The requested IP interface was not found.</td>
</tr>
<tr>
<td>702</td>
<td>Interface Not Resettable</td>
<td>One or more of the requested IP interfaces has a static address and cannot be reset (InterfaceReset).</td>
</tr>
<tr>
<td>703</td>
<td>Test Already Active</td>
<td>A test of this type is already active (and the implementation doesn’t support multiple active instances of a given test type).</td>
</tr>
<tr>
<td>704</td>
<td>Capabilities Preclude Test</td>
<td>Test arguments are individually valid but, taken together, describe a test that is beyond the service’s capabilities.</td>
</tr>
<tr>
<td>705</td>
<td>State Precludes Test</td>
<td>Service state precludes performing this test.</td>
</tr>
<tr>
<td>706</td>
<td>No Such Test</td>
<td>No test with the specified TestID was found.</td>
</tr>
<tr>
<td>707</td>
<td>Wrong Test Type</td>
<td>TestID is valid but refers to a different test type.</td>
</tr>
<tr>
<td>708</td>
<td>Invalid Test State</td>
<td>The TestID is valid but test results are not available.</td>
</tr>
<tr>
<td>709</td>
<td>State Precludes Cancel</td>
<td>The TestID is valid but the test can’t be canceled.</td>
</tr>
<tr>
<td>710</td>
<td>No Such Log</td>
<td>No log with the specified LogURI was found.</td>
</tr>
<tr>
<td>errorCode</td>
<td>errorDescription</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>711</td>
<td>Log Not Configurable</td>
<td>Log doesn’t permit enable/disable and/or log level to be changed.</td>
</tr>
<tr>
<td>712</td>
<td>Invalid Test Endpoint</td>
<td>The service does not support this test for the specified endpoint.</td>
</tr>
<tr>
<td>800-899</td>
<td>TBD</td>
<td><em>(Specified by UPnP vendor.)</em></td>
</tr>
</tbody>
</table>
2.6. Bandwidth Tests (Normative)

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:

- \textit{GetBandwidthTestInfo()} returns a \textit{BandwidthTestInfo} XML document.
- \textit{BandwidthTest()} supplies a \textit{BandwidthTestSpec} 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. Approach

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 \textit{italics}.
- There are quite a lot of possibilities, so for ease of reading the most commonly used terms are shown in \textbf{bold}.

\begin{verbatim}
BandwidthTestInfo ::= ( ProtocolKey + Endpoint + EndpointInfo )+
BandwidthTestSpec ::= ProtocolKey + SpecSettings
                     /* Endpoint is omitted here, because the same test spec applies to both server and client */
BandwidthTestResult ::= ProtocolKey + SpecSettings +
                      ( Endpoint + ResultKey + Results )*
                      /* note that this can include both server and client results */
ProtocolKey ::= Protocol + Version? + Profile
               /* protocol name, version and profile are defined in section 2.3.x */
Protocol ::= /* protocol name, e.g. "HTTP" or "Iperf" */
Endpoint ::= /* Server or Client */
EndpointInfo ::= SettingsCaps + FilesCaps + ResultsCaps +
               Instance*
SettingsCaps ::= SettingCap(Settings)
\end{verbatim}
/* i.e. “capabilities for all Settings” */

| SettingCap (Numeric-Setting) | ::= Settable? + AllowedValueRange? + AllowedValueList? |
| SettingCap (String-Setting) | ::= Settable? + AllowedValueList? |
| SettingCap (Boolean-Setting) | ::= Settable? + AllowedValueList? |
| Settable | ::= /* whether control point can set it; see section 2.6.x */ |
| AllowedValueRange | ::= Minimum + Maximum + Step? |
| Minimum | ::= /* minimum (inclusive) value */ |
| Maximum | ::= /* maximum (inclusive) value */ |
| Step | ::= /* possible values are {min, min+step, ...}; defaults to 1 */ |
| AllowedValueList | ::= AllowedValue+ |
| AllowedValue | ::= /* allowed value */ |

**FilesCaps** | ::= File*

**File** | ::= Path + RealFile? + SizeBytes? + MIMEType? /* individual file attributes are not defined in the EBNF */

**ResultsCaps** | ::= ResultCap (Results) /* “capabilities for all Results” */

**ResultCap (Result)** | ::= /* no additional info (indicates only that the result is supported) */

**SpecSettings** | ::= Settings + ( Endpoint + Settings )* 

**Instance** | ::= Settings 


**ResultKey** | ::= IntervalRange? + Stream? 

**IntervalRange** | ::= /* string of form “[start:end]” where start and end are milliseconds since start of test */

**Stream** | ::= /* opaque stream number, counting from zero */

**Results** | ::= TCPSYNTTime? + TCPSYNACKTime? + TCPLateACKs? + ROMTime? + BOMTime? + EOMTime? + TestBytesSent? + TestBytesReceived? + TotalBytesSent? + TotalBytesReceived? + TestPacketsSent? + TestPacketsReceived? + TotalPacketsSent? + TotalPacketsReceived?

/* values of results; individual settings are not defined in the EBNF; all are optional
(requirements are protocol-specific) */

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)
```

**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.

```
<xml version="1.0" encoding="UTF-8">
```

Copyright UPnP Forum © 2012. All rights reserved.
<bms:BandwidthTestResult xmlns:bms="urn:schemas-upnp-org:dm:bms"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

<!-- Protocol -->
<Protocol Name="HTTP" Profile="Baseline">

<!-- Settings -->
<Spec>
  <Host>192.168.1.1</Host>
  <TLSPort>443</TLSPort>
  <Direction>Get</Direction>
</Spec>

<!-- Endpoint -->
<Server>

<!-- Results -->
<ROMTime>2008-04-09T15:01:05.123456</ROMTime>
<BOMTime>2008-04-09T15:01:06.123456</BOMTime>
<EOMTime>2008-04-09T15:01:07.123456</EOMTime>
>TotalBytesReceived>123456</TotalBytesReceived>
</Server>
</Protocol>
</bms:BandwidthTestResult>

### 2.6.2. Protocol

| 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.

**Table 2-68: Bandwidth Test Protocols**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol</td>
<td>xsd:token</td>
<td>Required protocol name. The following protocol names are defined in this specification.</td>
<td>&quot;-&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>HTTP</strong>: RFC 2616 [HTTP]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>FTP</strong>: RFC 959 [FTP]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Iperf</strong>: [IPERF]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Echo</strong>: RFC 862 [ECHO]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>EchoPlus</strong>: [TR-143]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>

| Version   | xsd:token | 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+)* | "**" |

Copyright UPnP Forum © 2012. All rights reserved.
### 2.6.3. Capabilities

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.

#### Table 2-69: Bandwidth Test Capabilities

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Settings</td>
<td>Various</td>
<td>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.</td>
<td>Various</td>
</tr>
<tr>
<td>File[*]/Path</td>
<td>xsd:string</td>
<td>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.</td>
<td>-</td>
</tr>
<tr>
<td>File[*]/RealFile</td>
<td>xsd:boolean</td>
<td>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.</td>
<td>1</td>
</tr>
<tr>
<td>File[*]/SizeBytes</td>
<td>xsd:unsigned Long</td>
<td>Optional file size in bytes. Only applies to real files.</td>
<td>-</td>
</tr>
</tbody>
</table>
### Parameter Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>File[*]/MimeType</td>
<td>xsd:token</td>
<td>Optional file MIME type.</td>
<td>-</td>
</tr>
<tr>
<td>Results (TCPSYN-Time, TCPSYNACK-Time, TCPLateACKs etc)</td>
<td>Various</td>
<td>Results capabilities only indicate which results are supported. Individual results are defined in section 2.6.x.</td>
<td>n/a</td>
</tr>
</tbody>
</table>

### 2.6.4. Settings

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.

#### Table 2-70: Bandwidth Test Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface</td>
<td>xsd:token</td>
<td>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.</td>
<td>-</td>
</tr>
<tr>
<td>Transport</td>
<td>xsd:token</td>
<td>Layer 4 (transport) protocol. This specification only considers TCP and UDP, but additional transport protocols MAY be specified using their standard acronyms.</td>
<td>-</td>
</tr>
<tr>
<td>Host</td>
<td>xsd:token</td>
<td>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.</td>
<td>-</td>
</tr>
<tr>
<td>Port TLSPort</td>
<td>xsd:unsigned Short</td>
<td>Port number. For both client and server, this is a port on which the program will listen and accept connections or traffic.</td>
<td>-</td>
</tr>
<tr>
<td>SockBuffBytes</td>
<td>xsd:unsigned Int</td>
<td>Socket buffer size in bytes. Set using the setsockopt(SO_SNDBUF/SO_RCVBUF) system call. For TCP this is the TCP window.</td>
<td>-</td>
</tr>
<tr>
<td>AppBuffBytes</td>
<td>xsd:unsigned Int</td>
<td>Application buffer size in bytes. This is the length of the buffer that is used for read()/write() system calls.</td>
<td>-</td>
</tr>
</tbody>
</table>

Copyright UPnP Forum © 2012. All rights reserved.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Direction</strong></td>
<td>xsd:token</td>
<td>The direction, from the client point of view, in which test data will travel. Can be:</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Put</strong>: data will be sent from client to server.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Get</strong>: data will be sent from server to client.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>PutEcho</strong>: data will be sent from client to server, then echoed back to client.</td>
<td></td>
</tr>
<tr>
<td><strong>TimeMSecs</strong></td>
<td>xsd:unsigned Int</td>
<td>Determines the length of the test:</td>
<td>-</td>
</tr>
<tr>
<td><strong>LengthBytes</strong></td>
<td>xsd:unsigned Long</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>TimeMSecs</strong>: the length of time in milliseconds.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>LengthBytes</strong>: the number of bytes of test data that will be transferred.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not more than one of <strong>TimeMSecs</strong> and <strong>LengthBytes</strong> can be specified. If neither is specified, <strong>Path</strong> MUST be specified, and implicitly determines <strong>LengthBytes</strong>.</td>
<td></td>
</tr>
<tr>
<td><strong>Username</strong></td>
<td>xsd:string</td>
<td>The credentials with which a client authenticates itself to a server.</td>
<td>-</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>xsd:string</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This might be extended, e.g. to support certificates, in a future version.</td>
<td></td>
</tr>
<tr>
<td><strong>Path</strong></td>
<td>xsd:string</td>
<td>A relative path that identifies a data source or sink.</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See the definition of the <strong>Path</strong> capability in section 2.6.3.</td>
<td></td>
</tr>
<tr>
<td><strong>BandwidthBytes-PerSec</strong></td>
<td>xsd:unsigned Int</td>
<td>The desired bandwidth in bytes per second. If this is not specified, data will be sent as fast as possible.</td>
<td>-</td>
</tr>
<tr>
<td><strong>DSCP</strong></td>
<td>xsd:unsigned Int[0:63]</td>
<td>The Diffserv code point for marking packets transmitted in the test.</td>
<td>0</td>
</tr>
<tr>
<td><strong>EthernetPriority</strong></td>
<td>xsd:unsigned Int[0:7]</td>
<td>The Ethernet priority code for marking packets transmitted in the test.</td>
<td>0</td>
</tr>
<tr>
<td><strong>IntervalMSecs</strong></td>
<td>xsd:unsigned Int</td>
<td>The interval in milliseconds over which results are collected. For example, if <strong>TimeMSecs</strong> is 10000 and <strong>IntervalMSecs</strong> is 2000, the results for intervals [0:2000], [2000:4000], [4000:6000], [6000:8000]</td>
<td>-</td>
</tr>
<tr>
<td>Parameter</td>
<td>Type</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Threads</td>
<td>xsd:unsigned Int</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>TCPNoDelay</td>
<td>xsd:boolean</td>
<td>Applies only to TCP. Controls whether to disable Nagle’s algorithm, which is done using setsockopt(TCP_NODELAY).</td>
<td>0</td>
</tr>
<tr>
<td>TCPMSS</td>
<td>xsd:unsigned Int</td>
<td>Applies only to TCP. Controls the maximum segment size, which is done using setsockopt(TCP_MAXSEG).</td>
<td></td>
</tr>
<tr>
<td>ExitWhenDone</td>
<td>xsd:boolean</td>
<td>A non-binding request to a server or client to exit once a test has completed.</td>
<td>0</td>
</tr>
</tbody>
</table>

### 2.6.5. Results

\[ \text{Results} ::= \text{TCPSYNTIME}? + \text{TCPSYNACKTIME}? + \text{TCPLATEACKS}? + \text{ROMTIME}? + \text{BOMTIME}? + \text{EOMTIME}? + \text{TESTBYTESSENT}? + \text{TESTBYTESRECEIVED}? + \text{TOTALBYTESSENT}? + \text{TOTALBYTESRECEIVED}? + \text{TESTPACKETSENT}? + \text{TESTPACKETRECEIVED}? + \text{TOTALPACKETSENT}? + \text{TOTALPACKETRECEIVED}? + \text{LATEPACKETS}? + \text{DUPACKETS}? + \text{LOSTPACKETS}? + \text{OUTOFSERIALPACKETS}? + \text{RESENTPACKETS}? + \text{LATENCYUSECS}? + \text{JITTERUSECS}? \]

All results are optional but profiles can add requirements.

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

**Table 2-71: Bandwidth Test Results**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCPSYNTIME</td>
<td>xsd:dateTime</td>
<td>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.</td>
</tr>
<tr>
<td>TCPSYNACKTIME</td>
<td>xsd:dateTime</td>
<td>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.</td>
</tr>
<tr>
<td>TCPLATEACKS</td>
<td>xsd:unsigned Int</td>
<td>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 <a href="http://www.stuartcheshire.org/papers/Nagle-DelayedAck">http://www.stuartcheshire.org/papers/Nagle-DelayedAck</a> provides useful background discussion and suggests that a late ACK might be defined as one that takes over 100ms.</td>
</tr>
<tr>
<td>ROMTIME</td>
<td>xsd:dateTime</td>
<td>The absolute time at which the client sent (or the server received) the first protocol-specific request on the</td>
</tr>
<tr>
<td>Parameter</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>BOMTime</strong></td>
<td>xsd:dateTime</td>
<td>The absolute time at which the client (or server) first knew that test data transmission could begin. This is worded carefully to cover protocols where there is no initial negotiation (e.g. HTTP, Echo) and protocols where there is an initial negotiation (e.g. FTP).</td>
</tr>
<tr>
<td><strong>EOMTime</strong></td>
<td>xsd:dateTime</td>
<td>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 there is such an explicit indication (e.g. FTP).</td>
</tr>
<tr>
<td><strong>TestBytesSent</strong></td>
<td>xsd:unsigned Long</td>
<td>The number bytes of test data that were transferred over the data connection.</td>
</tr>
<tr>
<td><strong>TestBytesReceived</strong></td>
<td>xsd:unsigned Long</td>
<td>The total number of IP payload bytes that were transferred over the data connection.</td>
</tr>
<tr>
<td><strong>TotalBytesSent</strong></td>
<td>xsd:unsigned Long</td>
<td>The number of protocol-specific packets that were transferred over the data connection.</td>
</tr>
<tr>
<td><strong>TotalBytesReceived</strong></td>
<td>xsd:unsigned Long</td>
<td>The number of IP packets that were transferred over the data connection.</td>
</tr>
<tr>
<td><strong>TestPacketsSent</strong></td>
<td>xsd:unsigned Int</td>
<td>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.</td>
</tr>
<tr>
<td><strong>TestPacketsReceived</strong></td>
<td>xsd:unsigned Int</td>
<td>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”.</td>
</tr>
<tr>
<td><strong>DuplicatePackets</strong></td>
<td>xsd:unsigned Int</td>
<td>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”. 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”.</td>
</tr>
<tr>
<td><strong>LostPackets</strong></td>
<td>xsd:unsigned Int</td>
<td>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 “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”.</td>
</tr>
<tr>
<td><strong>OutOfSequencePackets</strong></td>
<td>xsd:unsigned Int</td>
<td>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 “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”.</td>
</tr>
<tr>
<td><strong>ResentPackets</strong></td>
<td>xsd:unsigned Int</td>
<td>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</td>
</tr>
</tbody>
</table>
Parameter | Type       | Description
----------|------------|------------------------
           |            | same packet is re-sent \(n\) times then the counter is incremented by \(n\).

**LatencyUSecs**

| xsd:unsigned Int | 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**

| xsd:int       | 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. Profiles

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 **Setting/Result** column, which indicates whether the parameter is a setting (S) or a result (R).

For all settings (S) 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 (R) listed in the table:

---

Copyright UPnP Forum © 2012. All rights reserved.
• 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.

Table 2-72: Bandwidth Test HTTP and FTP Baseline Profile

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting/Result</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint</td>
<td></td>
<td>MUST support Server and/or Client</td>
</tr>
<tr>
<td>Transport</td>
<td>S</td>
<td>MUST support TCP</td>
</tr>
<tr>
<td>Host</td>
<td>S</td>
<td>A value MUST be supplied in BandwidthTestSpec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This is always the host to which the client will</td>
</tr>
<tr>
<td></td>
<td></td>
<td>connect</td>
</tr>
<tr>
<td>Port</td>
<td>S</td>
<td>This is always the port to which the client will</td>
</tr>
<tr>
<td></td>
<td></td>
<td>connect</td>
</tr>
<tr>
<td>Direction</td>
<td>S</td>
<td>MUST support Get (download) and/or Put (upload)</td>
</tr>
<tr>
<td>LengthBytes</td>
<td>S</td>
<td>MUST be supported if Direction Put (upload) is supported</td>
</tr>
<tr>
<td>Path</td>
<td>S</td>
<td>A value MUST be supplied in BandwidthTestSpec</td>
</tr>
<tr>
<td>TotalBytesSent</td>
<td>R</td>
<td>MUST be supported by Client if Direction Put (upload) is supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MUST be supported by Server if Direction Get (download) is supported</td>
</tr>
<tr>
<td>TotalBytesReceived</td>
<td>R</td>
<td>MUST be supported by Client if Direction Get (download) is supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MUST be supported by Server if Direction Put (upload) is supported</td>
</tr>
</tbody>
</table>

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

Table 2-73: Bandwidth Test HTTP and FTP BBF Profile

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting/Result</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint</td>
<td></td>
<td>MUST support Client</td>
</tr>
<tr>
<td>Parameter</td>
<td>Setting/Result</td>
<td>Comment</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Protocol</td>
<td></td>
<td>If <strong>HTTP</strong> is supported, persistent connections MUST be used, pipelining MUST NOT be used, and HTTP authentication MUST NOT be used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If <strong>FTP</strong> is supported, binary transfers MUST be used.</td>
</tr>
<tr>
<td>Transport</td>
<td>S</td>
<td>MUST support <strong>TCP</strong>.</td>
</tr>
<tr>
<td>Host</td>
<td>S</td>
<td>A value MUST be supplied in <strong>BandwidthTestSpec</strong>. This is always the host to which the client will connect.</td>
</tr>
<tr>
<td>Port</td>
<td>S</td>
<td>This is always the port to which the client will connect.</td>
</tr>
<tr>
<td>Direction</td>
<td>S</td>
<td>MUST support <strong>Get</strong> (download) and/or <strong>Put</strong> (upload).</td>
</tr>
<tr>
<td>LengthBytes</td>
<td>S</td>
<td>MUST be supported if <strong>Direction Put</strong> (upload) is supported.</td>
</tr>
<tr>
<td>Path</td>
<td>S</td>
<td>A value MUST be supplied in <strong>BandwidthTestSpec</strong>.</td>
</tr>
<tr>
<td>DSCP</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>EthernetPriority</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>TCPSYNT ime</td>
<td>R</td>
<td>(TR-143 parameter is TCPOpenRequestTime) MAY be supported; if supported, MUST behave as specified:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For <strong>HTTP</strong> this is the time at which the TCP socket open (SYN) was sent for the HTTP connection.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For <strong>FTP</strong> this is the time at which the TCP socket open (SYN) was sent for the data connection.</td>
</tr>
<tr>
<td>TCPSYNACKTime</td>
<td>R</td>
<td>(TR-143 parameter is TCPOpenResponseTime) MAY be supported; if supported, MUST behave as specified:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For <strong>HTTP</strong> this is the time at which the TCP ACK to the socket opening the HTTP connection was received.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For <strong>FTP</strong> this is the time at which the TCP ACK to the socket opening the Data connection was received.</td>
</tr>
<tr>
<td>ROMTime</td>
<td>R</td>
<td>For <strong>HTTP</strong> this is the time at which the client sends the GET / PUT command.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For <strong>FTP</strong> this is the time at which the client sends the RTRV / STOR command.</td>
</tr>
<tr>
<td>BOMTime</td>
<td>R</td>
<td>For <strong>HTTP</strong> this is the time at which the first data packet is received / sent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For <strong>FTP</strong> this is the time at which the client receives the first data packet on the data connection (<strong>Get</strong>) or the ready for transfer notification (<strong>Put</strong>)</td>
</tr>
<tr>
<td>EOMTime</td>
<td>R</td>
<td>For <strong>HTTP</strong> this is the time at which the last data packet (<strong>Get</strong>) or successful response code (<strong>Put</strong>) is received.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For <strong>FTP</strong> this is the time at which the client receives the last packet on the data connection (<strong>Get</strong>) or a transfer complete notification (<strong>Put</strong>)</td>
</tr>
<tr>
<td>Parameter</td>
<td>Setting/Result</td>
<td>Comment</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TestBytesReceived</td>
<td>R</td>
<td>MUST be supported by Client if Direction Get (download) is supported and 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.</td>
</tr>
<tr>
<td>TotalBytesSent</td>
<td>R</td>
<td>MUST be supported by Client if Direction Put (upload) is supported and by Server if Direction Get (download) is supported. The total number of bytes sent on the interface between BOMTime and EOMTime.</td>
</tr>
<tr>
<td>TotalBytesReceived</td>
<td>R</td>
<td>MUST be supported by Client if Direction Get (download) is supported and by Server if Direction Put (upload) is supported. The total number of bytes received on the interface between BOMTime and EOMTime.</td>
</tr>
</tbody>
</table>

### 2.6.6.2. Echo and EchoPlus Profiles

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

#### Table 2-74: Bandwidth Test Echo and EchoPlus Baseline Profile

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting/Result</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint</td>
<td></td>
<td>MUST support Server.</td>
</tr>
<tr>
<td>Protocol</td>
<td></td>
<td>GetBandwidthTestResult() MUST support the return of partial results while the test is still active.</td>
</tr>
<tr>
<td>Transport</td>
<td>S</td>
<td>MUST support UDP.</td>
</tr>
<tr>
<td>Port</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>Direction</td>
<td>S</td>
<td>MUST support PutEcho. This is implied by the protocol, so it MAY be omitted from BandwidthTestInfo, BandwidthTestSpec and BandwidthTestResult.</td>
</tr>
<tr>
<td>TotalBytesSent</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TotalBytesReceived</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

These requirements (a superset of the Baseline requirements) apply when Protocol is Echo or EchoPlus and Profile is BBF.

#### Table 2-75: Bandwidth Test Echo and EchoPlus BBF Profile

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting/Result</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint</td>
<td></td>
<td>MUST support Server.</td>
</tr>
<tr>
<td>Protocol</td>
<td></td>
<td>GetBandwidthTestResult() MUST support the return of intermediate results while the test is still active.</td>
</tr>
</tbody>
</table>
### 2.6.6.3. Iperf profiles

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

Table 2-76: Bandwidth Test Iperf Baseline Profile

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting/Result</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint</td>
<td></td>
<td>MUST support Server and/or Client</td>
</tr>
<tr>
<td>Transport</td>
<td>S</td>
<td>MUST support TCP and UDP</td>
</tr>
<tr>
<td>Host</td>
<td>S</td>
<td>A value MUST be supplied in BandwidthTestSpec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If within a &lt;Server&gt; XML element, this determines the IP address to which the server will bind; otherwise it is the host to which the client will connect</td>
</tr>
<tr>
<td>Port</td>
<td>S</td>
<td>If within a &lt;Client&gt; 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</td>
</tr>
<tr>
<td>SockBuffBytes</td>
<td>S</td>
<td>Only used by client; MUST support Put and PutEcho</td>
</tr>
<tr>
<td>AppBuffBytes</td>
<td>S</td>
<td>Only used by client; MUST support Put and PutEcho</td>
</tr>
<tr>
<td>Direction</td>
<td>S</td>
<td>Only used by client; MUST support Put and PutEcho</td>
</tr>
</tbody>
</table>
### Parameter Setting/Result Comment

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting/Result</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeMSecs</td>
<td>S</td>
<td>Only used by client</td>
</tr>
<tr>
<td>LengthBytes</td>
<td>S</td>
<td>Only used by client</td>
</tr>
<tr>
<td>BandwidthBytesPerSec</td>
<td>S</td>
<td>Only used by client; only applies to <strong>UDP</strong> tests</td>
</tr>
<tr>
<td>IntervalMSecs</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>Threads</td>
<td>S</td>
<td>Defaults to Iperf default</td>
</tr>
<tr>
<td>TCPNoDelay</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>TCPMSS</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>ExitWhenDone</td>
<td>S</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TestBytesSent TestBytesReceived</th>
<th>R</th>
<th>Only applies to <strong>UDP</strong> tests</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>TestPacketsSent TestPacketsReceived</th>
<th>R</th>
<th>Only applies to <strong>UDP</strong> tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>LostPackets</td>
<td>R</td>
<td>Only applies to <strong>UDP</strong> tests</td>
</tr>
<tr>
<td>JitterUSecs</td>
<td>R</td>
<td>Only applies to <strong>UDP PutEcho</strong> tests</td>
</tr>
</tbody>
</table>

### Extensibility

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. Theory of Operation (Informative)

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

2.7.1. Assumptions

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

**Figure 2-3: Example 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 /UPnP/DM/DeviceInfo/OperatingSystem/WillReboot and /UPnP/DM/DeviceInfo/OperatingSystem/WillBaselineReset are "1", 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.

  - There is a single IP interface which (obviously) is used for UPnP management.
  - The implementation can execute a maximum of one test of each type at a time.
  - There is a single log file, which includes OS-level messages and UPnP action-oriented messages.
  - The *DeviceProtection:1 Role Lists* and *Restricted Role Lists* are as listed in Table 2-15, and the *BasicManagement:2* access control list is as listed in the section 2.3.37 example.

### 2.7.1.2. *Parent Device* #2a

Some of the examples also consider the more complicated *Parent Device* #2a, which differs from *Parent Device* #1 in the following respects:

- The targeted EE is the JVM, so any EE-related actions and data model apply to the JVM.

- Both /UPnP/DM/DeviceInfo/ExecutionEnvironment/WillReboot and /UPnP/DM/DeviceInfo/ExecutionEnvironment/WillBaselineReset are “1”, and both /UPnP/DM/DeviceInfo/OperatingSystem/WillReboot and /UPnP/DM/DeviceInfo/OperatingSystem/WillBaselineReset are “0”, so the *Reboot()* and *BaselineReset()* actions apply to the targeted EE (JVM) but not to the OS. Therefore:
  - Rebooting the *Parent Device* causes a complete restart of the *Parent Device*, its embedded devices / services, and the targeted EE (JVM).
  - Resetting the *Parent Device* to its baseline state causes a complete restart of the *Parent Device*, its embedded devices / services, and the targeted EE (JVM). Persistent settings revert to their original values.

### 2.7.1.3. *Parent Device* #2b

Some of the examples also consider *Parent Device* #2b, which is an alternative to *Parent Device* #2a and differs from it in the following respects:

- It is implemented using JVM services and doesn’t have access to anything outside the JVM.

- Rebooting the *Parent Device* causes a complete restart of the EE (JVM).

- Resetting the *Parent Device* to its baseline state causes a complete restart of the EE (JVM). Persistent settings revert to their baseline values.

- There are two log files, one which includes UPnP action-oriented messages and another which contains EE (JVM) messages.

### 2.7.2. Rebooting the *Parent Device*

The *Reboot()* action (section 2.5.1) 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.

Consider several scenarios:

- The device implementation is able to reboot immediately, in which case the action completes successfully and returns a *RebootStatus* value of *RebootNow*.
• 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 “1” 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 `ssdp:byebye` 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`.

![Figure 2-4: RebootNow Example](image-url)
2.7.3. Resetting the 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 “1” 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 `ssdp:byebye` 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 “1” and each `ssdp:byebye` 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:
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. Using Sequence Mode

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 “0”.
- Control point A calls `SetSequenceMode("1")`, discovering that it was previously “0”, and therefore knowing that it can proceed to make its changes.
- Control point B calls `SetSequenceMode("1")`, discovering that it was previously “1”, 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("0")`, indicating that it has finished making its changes.
- Control point B discovers (via polling or eventing) that `SequenceMode` is now “0”, so proceeds to set it to “1”, make its changes, and set it back to “0”.

---

*Copyright UPnP Forum © 2012. All rights reserved.*
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("0")`, either control point B or control point C would have managed to set `SequenceMode` to “1” 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 “0”.
- Control point A calls `SetSequenceMode("1")`, discovering that it was previously “0”, and therefore knowing that it can proceed to make its changes.
- Control point B calls `SetSequenceMode("1")`, discovering that it was previously “1”, and therefore knowing that it can’t proceed to make its changes.
- Control point A makes its changes. `SequenceMode` is “1”, 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("0")`, 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 “0”, so proceeds to set it to “1”, make its changes, and set it back to “0”.

Figure 2-6 below illustrates the use of `SequenceMode`. 
Figure 2-6: **SequenceMode** Example
2.7.5. Running a Ping Test
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) : 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) \(\Rightarrow (\text{“Success”}, \text{“”}, 9, 1, 45, 40, 50)\) 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) \(\Rightarrow (\text{“Error_CannotResolveHostName”}, \text{“Timeout”}, 0, 0, 0, 0, 0)\) 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. Running an NSLookup Test
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:
BasicManagement:2 Service Template Version 1.01

- **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. Running a Traceroute Test

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 \texttt{GetTracerouteResult(44)} to retrieve the test results. For example:
  
  \begin{itemize}
    \item \texttt{GetTracerouteResult(44)} $\rightarrow$ \(("Success", ",", 888, "1.2.3.4,2.3.4.5,4.5.6.7")\) 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.
    \item \texttt{GetTracerouteResult(44)} $\rightarrow$ \(("Error\_MaxHopCountExceeded", "Timeout", 0, ",")\) indicates that the trace-route test failed because the number of hops to www.myserver.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).
  \end{itemize}

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

2.7.8. Running an InterfaceReset Test
The \texttt{InterfaceReset()} and \texttt{GetInterfaceResetResult()} actions (sections 2.5.15 and 2.5.16) are optional.

• \texttt{InterfaceReset()}’s Role List is “\texttt{Admin}” and its Recommended Role List is “\texttt{Basic}”. This means that control points that possess the \texttt{Admin Role} are unconditionally permitted to invoke the action. A control point that possesses only the \texttt{Basic Role} is (as explained in section 2.5.15.2) only permitted to invoke the action if the value of the \texttt{Interfaces} argument is “\texttt{RequestInterface}” (because we are assuming that the \texttt{BasicManagement:2} access control list is as listed in the section 2.3.37 example).

• \texttt{GetInterfaceResetResult()}’s Role List is “\texttt{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 \texttt{BasicManagement} events and discovers that \texttt{ActiveTestIDs} is “\texttt{"}”, indicating that no tests are currently active.

• Control point calls \texttt{InterfaceReset()}. For example:
  
  \begin{itemize}
    \item \texttt{InterfaceReset(\text{"AllInterfaces\") :}} reset all IP interfaces. This is permitted only if the control point possesses the \texttt{Admin Role}; for other control points, the action would fail with a 606 (Action not authorized) error.
    \item \texttt{InterfaceReset(\text{"RequestInterface\") :}} reset the IP interface on which the action request was received. This is permitted only if the control point possesses the \texttt{Admin or Basic Role}; for other control points, the action would fail with a 606 (Action not authorized) error.
    \item \texttt{InterfaceReset(\text{"X_UPNP\_ORG\_lan\") :}} reset the IP interface whose system name is \texttt{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 \texttt{Admin Role}; for other control points, the action would fail with a 606 (Action not authorized) error.
  \end{itemize}

• Control point receives test ID (46 for example) in the \texttt{InterfaceReset()} response. It also discovers, via an event, that \texttt{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 \texttt{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:
  o 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.
  o 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. Running a Self Test

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

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:

**Table 2-77: Bandwidth Test Discovered Devices**

<table>
<thead>
<tr>
<th>ID</th>
<th>Protocol</th>
<th>Version</th>
<th>Profile</th>
<th>Endpoint</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HTTP</td>
<td>1.1</td>
<td>Baseline</td>
<td>Server</td>
<td>Get</td>
</tr>
<tr>
<td>2</td>
<td>HTTP</td>
<td>1.1</td>
<td>Baseline</td>
<td>Client</td>
<td>Get</td>
</tr>
<tr>
<td>3</td>
<td>echo</td>
<td></td>
<td>Baseline</td>
<td>Server</td>
<td>PutEcho</td>
</tr>
<tr>
<td>4</td>
<td>Iperf</td>
<td>2.0.4</td>
<td>Baseline</td>
<td>Server</td>
<td>PutEcho</td>
</tr>
<tr>
<td>5</td>
<td>Iperf</td>
<td>3.0.0</td>
<td>Baseline</td>
<td>Server</td>
<td>PutEcho</td>
</tr>
<tr>
<td>6</td>
<td>Iperf</td>
<td>2.0.5</td>
<td>Baseline</td>
<td>Client</td>
<td>PutEcho</td>
</tr>
</tbody>
</table>

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:
  o 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:
  o 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:
  o 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).
  o GetBandwidthTestResult(45)  (“Error_InitConnectionFailed”, “Timeout”, “”) indicates that the bandwidth test failed to connect to www.myserver.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.

Copyright UPnP Forum © 2012. All rights reserved.
Here are BandwidthTestInfo, BandwidthTestSpec and BandwidthTestResult XML documents that illustrate this use case.

Row 1 (HTTP server) BandwidthTestInfo

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestInfo xmlns:bms="urn:schemas-upnp-org:dm:bms"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:schemas-upnp-org:dm:bms
   http://www.upnp.org/schemas/dm/bms.xsd">
...
<Protocol Name="HTTP" Version="1.1" Profile="Baseline">
  <Server>
    <Transport>
      <AllowedValueList>
        <AllowedValue>TCP</AllowedValue>
      </AllowedValueList>
    </Transport>
    <Host/>
    <Port>
      <AllowedValueList>
        <AllowedValue>80</AllowedValue>
      </AllowedValueList>
    </Port>
    <Direction>
      <AllowedValueList>
        <AllowedValue>Get</AllowedValue>
      </AllowedValueList>
    </Direction>
    <Path/>
    <TotalBytesSent/>
  </Server>
</Protocol>
...
</bms:BandwidthTestInfo>
```

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestInfo xmlns:bms="urn:schemas-upnp-org:dm:bms"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:schemas-upnp-org:dm:bms
   http://www.upnp.org/schemas/dm/bms.xsd">
<Protocol Name="HTTP" Version="1.1" Profile="Baseline">
  <Client>
    <Transport>
      <AllowedValueList>
        <AllowedValue>TCP</AllowedValue>
      </AllowedValueList>
    </Transport>
    <Host/>
    <Port/>
    <Direction>
      <AllowedValueList>
        <AllowedValue>Get</AllowedValue>
      </AllowedValueList>
    </Direction>
    <Path/>
    <TotalBytesSent/>
  </Client>
</Protocol>
```
<Path/>
<TotalBytesReceived/>
</Client>
</Protocol>

</bms:BandwidthTestInfo>

BandwidthTestSpec (spec is the same for client and server)

<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestSpec xmlns:bms="urn:schemas-upnp-org:dm:bms"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:schemas-upnp-org:dm:bms
http://www.upnp.org/schemas/dm/bms.xsd">
  <Protocol Name="HTTP" Version="1.1" Profile="Baseline">
    <Spec>
      <Transport>TCP</Transport>
      <Host>192.168.1.1</Host>
      <Port>80</Port>
      <Direction>Get</Direction>
      <Path>data/test.dat</Path>
    </Spec>
    <Server>
      <TotalBytesSent>123456</TotalBytesSent>
    </Server>
  </Protocol>
</bms:BandwidthTestSpec>

Row 1 (HTTP server) BandwidthTestResult

<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestResult xmlns:bms="urn:schemas-upnp-org:dm:bms"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:schemas-upnp-org:dm:bms
http://www.upnp.org/schemas/dm/bms.xsd">
  <Protocol Name="HTTP" Version="1.1" Profile="Baseline">
    <Spec>
      <Transport>TCP</Transport>
      <Host>192.168.1.1</Host>
      <Port>80</Port>
      <Direction>Get</Direction>
      <Path>data/test.dat</Path>
    </Spec>
    <Server>
      <TotalBytesSent>123456</TotalBytesSent>
    </Server>
  </Protocol>
</bms:BandwidthTestResult>

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

<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestResult xmlns:bms="urn:schemas-upnp-org:dm:bms"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:schemas-upnp-org:dm:bms
http://www.upnp.org/schemas/dm/bms.xsd">
  <Protocol Name="HTTP" Version="1.1" Profile="Baseline">
    <Spec>
      <Transport>TCP</Transport>
      <Host>192.168.1.1</Host>
      <Port>80</Port>
      <Direction>Get</Direction>
      <Path>data/test.dat</Path>
    </Spec>
    <Server>
      <TotalBytesSent>123456</TotalBytesSent>
    </Server>
  </Protocol>
</bms:BandwidthTestResult>
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)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestInfo xmlns:bms="urn:schemas-upnp-org:dm:bms"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <Protocol Name="HTTP" Version="1.1" Profile="Baseline">
        ...
    </Protocol>
    <Protocol Name="HTTP" Version="1.1" Profile="BBF"
        BaseProfiles="Baseline">
        <Client>
            <Transport>
                <AllowedValueList>
                    <AllowedValue>TCP</AllowedValue>
                </AllowedValueList>
            </Transport>
            <Host/>
            <Port/>
            <Direction>
                <AllowedValueList>
                    <AllowedValue>Get</AllowedValue>
                </AllowedValueList>
            </Direction>
            <Path/>
            <DSCP/>
            <EthernetPriority/>
            <ROMTime/>
            <BOMTime/>
            <EOMTime/>
        </Client>
    </Protocol>
</bms:BandwidthTestInfo>
```
BandwidthTestSpec (differences from Baseline are shown in bold)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestSpec xmlns:bms="urn:schemas-upnp-org:dm:bms"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <Protocol Name="HTTP" Version="1.1" Profile="BBF"
        BaseProfiles="Baseline">
        <Transport>TCP</Transport>
        <Host>192.168.1.1</Host>
        <Port>80</Port>
        <Direction>Get</Direction>
        <Path>data/test.dat</Path>
    </Protocol>
</bms:BandwidthTestSpec>
```

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestResult xmlns:bms="urn:schemas-upnp-org:dm:bms"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <Protocol Name="HTTP" Version="1.1" Profile="BBF"
        BaseProfiles="Baseline">
        <Spec>
            <Transport>TCP</Transport>
            <Host>192.168.1.1</Host>
            <Port>80</Port>
            <Direction>Get</Direction>
            <Path>data/test.dat</Path>
            <DSCP>0</DSCP>
            <EthernetPriority>0</EthernetPriority>
        </Spec>
        <Client>
            <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>
            <TotalBytesReceived>123456</TotalBytesReceived>
        </Client>
    </Protocol>
</bms:BandwidthTestResult>
```

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**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestInfo xmlns:bms="urn:schemas-upnp-org:dm:bms"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <Protocol Name="Echo" Profile="Baseline">
        <Server>
            <Transport>
                <AllowedValueList>
                    <AllowedValue>UDP</AllowedValue>
                </AllowedValueList>
            </Transport>
            <Port>
                <AllowedValueList>
                    <AllowedValue>7</AllowedValue>
                </AllowedValueList>
            </Port>
            <TotalBytesSent/>
            <TotalBytesReceived/>
        </Server>
    </Protocol>
    <Protocol Name="Echo" Profile="BBF">
        ...
    </Protocol>
</bms:BandwidthTestInfo>
```

**BandwidthTestSpec**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestSpec xmlns:bms="urn:schemas-upnp-org:dm:bms"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <Protocol Name="Echo" Profile="Baseline">
        <Transport>UDP</Transport>
        <Port/>7</Protocol>
    </Protocol>
</bms:BandwidthTestSpec>
```

**Row 3 (Echo server) BandwidthTestResult**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestResult xmlns:bms="urn:schemas-upnp-org:dm:bms"
```
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**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestInfo xmlns:bms="urn:schemas-upnp-org:dm:bms"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:schemas-upnp-org:dm:bms
    http://www.upnp.org/schemas/dm/bms.xsd">
    <Protocol Name="Iperf" Version="2.0.4" Profile="Baseline">
        <Server>
            <Transport>
                <AllowedValueList>
                    <AllowedValue>TCP</AllowedValue>
                    <AllowedValue>UDP</AllowedValue>
                </AllowedValueList>
            </Transport>
            <Host/>
            <Port>
                <AllowedValueList>
                    <AllowedValue>5001</AllowedValue>
                </AllowedValueList>
            </Port>
            <SockBuffBytes/>
            <AppBuffBytes/>
            <IntervalMSecs/>
            <Threads/>
            <TCPNoDelay/>
        </Server>
    </Protocol>
</bms:BandwidthTestInfo>
```
<TCPMSS/>
<ExitWhenDone/>
<TestBytesSent/>
<TestBytesReceived/>
<TestPacketsSent/>
<TestPacketsReceived/>
<LostPackets/>
<JitterUSecs/>
</Server>
</Protocol>
</bms:BandwidthTestInfo>

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestInfo xmlns:bms="urn:schemas-upnp-org:dm:bms"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
<BandwidthTestInfo>
<Protocol Name="Iperf" Version="2.0.5" Profile="Baseline">
  <Client>
    <Transport>
      <AllowedValueList>
        <AllowedValue>TCP</AllowedValue>
        <AllowedValue>UDP</AllowedValue>
      </AllowedValueList>
    </Transport>
    <Host/>
    <Port>
      <AllowedValueList>
        <AllowedValue>5002</AllowedValue>
      </AllowedValueList>
    </Port>
    <SockBuffBytes Settable="1"/>
    <AppBuffBytes Settable="1"/>
    <Direction>
      <AllowedValueList>
        <AllowedValue>PutEcho</AllowedValue>
      </AllowedValueList>
    </Direction>
    <TimeMSecs/>
    <LengthBytes/>
    <BandwidthBytesPerSec/>
    <IntervalMSecs/>
    <Threads/>
    <TCPNoDelay/>
    <TCPMSS/>
    <ExitWhenDone/>
    <TestBytesSent/>
    <TestBytesReceived/>
    <TestPacketsSent/>
    <TestPacketsReceived/>
    <LostPackets/>
    <JitterUSecs/>
  </Client>
</Protocol>
</bms:BandwidthTestInfo>
```
BasicManagement:2 Service Template Version 1.0

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestSpec xmlns:bms="urn:schemas-upnp-org:dm:bms"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <Protocol Name="Iperf" Version="2" Profile="Baseline">
        <Transport>TCP</Transport>
        <Port>5001</Port>
        <ExitWhenDone>1</ExitWhenDone>
        <Client>
            <Host>192.168.1.1</Host>
            <Port>5002</Port>
            <Direction>PutEcho</Direction>
            <TimeMSecs>10000</TimeMSecs>
        </Client>
    </Protocol>
</bms:BandwidthTestSpec>
```

Row 4 (Iperf server) BandwidthTestResult

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestResult xmlns:bms="urn:schemas-upnp-org:dm:bms"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <Protocol Name="Iperf" Version="2.0.4" Profile="Baseline">
        <Spec>
            <Transport>TCP</Transport>
            <Port>5001</Port>
            <IntervalMSecs>2000</IntervalMSecs>
            <ExitWhenDone>1</ExitWhenDone>
            <Client>
                <Host>192.168.1.1</Host>
                <Port>5002</Port>
                <Direction>PutEcho</Direction>
                <TimeMSecs>10000</TimeMSecs>
                <Threads>2</Threads>
            </Client>
        </Spec>
    </Protocol>
</bms:BandwidthTestResult>
```

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bms:BandwidthTestResult xmlns:bms="urn:schemas-upnp-org:dm:bms"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <Server>
        <TestBytesSent>123456</TestBytesSent>
        <TestBytesReceived>123456</TestBytesReceived>
    </Server>
</bms:BandwidthTestResult>
```
<Protocol Name="Iperf" Version="2.0.5" Profile="Baseline">
  <Spec>
    <Transport>TCP</Transport>
    <Port>5001</Port>
    <Interval MSecs>2000</Interval MSecs>
    <ExitWhenDone>1</ExitWhenDone>
    <Client>
      <Host>192.168.1.1</Host>
      <Port>5002</Port>
      <Direction>PutEcho</Direction>
      <TimeMSecs>10000</TimeMSecs>
      <Threads>2</Threads>
    </Client>
  </Spec>
  <Client>
    <TestBytesSent>123456</TestBytesSent>
    <TestBytesReceived>123456</TestBytesReceived>
  </Client>
</Protocol>
</bms:BandwidthTestResult>

2.7.11. Manipulating Logs

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() \(\rightarrow\) (“urn:example-com:private:device-log”) : a single log for Parent Device #1 and #2a.

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”) \(\rightarrow\) (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”) \(\rightarrow\) (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)

```xml
<?xml version="1.0"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <actionList>
    <action>
      <name>Reboot</name>
      <argumentList>
        <argument>
          <name>RebootStatus</name>
          <direction>out</direction>
          <relatedStateVariable>A_ARG_TYPE_RebootStatus</relatedStateVariable>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>BaselineReset</name>
    </action>
    <action>
      <name>GetDeviceStatus</name>
      <argumentList>
        <argument>
          <name>DeviceStatus</name>
          <direction>out</direction>
          <relatedStateVariable>DeviceStatus</relatedStateVariable>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>SetSequenceMode</name>
      <argumentList>
        <argument>
          <name>NewSequenceMode</name>
          <direction>in</direction>
          <relatedStateVariable>SequenceMode</relatedStateVariable>
        </argument>
        <argument>
          <name>OldSequenceMode</name>
          <direction>out</direction>
          <relatedStateVariable>SequenceMode</relatedStateVariable>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>GetSequenceMode</name>
      <argumentList>
        <argument>
          <name>SequenceMode</name>
          <direction>out</direction>
          <relatedStateVariable>SequenceMode</relatedStateVariable>
        </argument>
      </argumentList>
    </action>
  </actionList>
</scpd>
```
<action>
    <name>Ping</name>
    <argumentList>
        <argument>
            <name>Host</name>
            <direction>in</direction>
            <relatedStateVariable>A_ARG_TYPE_Host</relatedStateVariable>
        </argument>
        <argument>
            <name>NumberOfRepetitions</name>
            <direction>in</direction>
            <relatedStateVariable>A_ARG_TYPE(UInt)</relatedStateVariable>
        </argument>
        <argument>
            <name>Timeout</name>
            <direction>in</direction>
            <relatedStateVariable>A_ARG_TYPE(MSecs)</relatedStateVariable>
        </argument>
        <argument>
            <name>DataBlockSize</name>
            <direction>in</direction>
            <relatedStateVariable>A_ARG_TYPE(UInt)</relatedStateVariable>
        </argument>
        <argument>
            <name>DSCP</name>
            <direction>in</direction>
            <relatedStateVariable>A_ARG_TYPE(DSCP)</relatedStateVariable>
        </argument>
        <argument>
            <name>TestID</name>
            <direction>out</direction>
            <relatedStateVariable>A_ARG_TYPE(TestID)</relatedStateVariable>
        </argument>
    </argumentList>
</action>

<action>
    <name>GetPingResult</name>
    <argumentList>
        <argument>
            <name>TestID</name>
            <direction>in</direction>
            <relatedStateVariable>A_ARG_TYPE(TestID)</relatedStateVariable>
        </argument>
        <argument>
            <name>Status</name>
            <direction>out</direction>
            <relatedStateVariable>A_ARG_TYPE(PingStatus)</relatedStateVariable>
        </argument>
        <argument>
            <name>AdditionalInfo</name>
            <direction>out</direction>
        </argument>
    </argumentList>
</action>
<relatedStateVariable>A_ARG_TYPE_String</relatedStateVariable>
</argument>
<argument>
  <name>SuccessCount</name>
  <direction>out</direction>
  <relatedStateVariable>A_ARG_TYPE_UInt</relatedStateVariable>
</argument>
<argument>
  <name>FailureCount</name>
  <direction>out</direction>
  <relatedStateVariable>A_ARG_TYPE_UInt</relatedStateVariable>
</argument>
<argument>
  <name>AverageResponseTime</name>
  <direction>out</direction>
  <relatedStateVariable>A_ARG_TYPE_MSecs</relatedStateVariable>
</argument>
<argument>
  <name>MinimumResponseTime</name>
  <direction>out</direction>
  <relatedStateVariable>A_ARG_TYPE_MSecs</relatedStateVariable>
</argument>
<argument>
  <name>MaximumResponseTime</name>
  <direction>out</direction>
  <relatedStateVariable>A_ARG_TYPE_MSecs</relatedStateVariable>
</argument>
</argumentList>
</action>
</action>
<name>NSLookup</name>
<argumentList>
<argument>
  <name>HostName</name>
  <direction>in</direction>
  <relatedStateVariable>A_ARG_TYPE_HostName</relatedStateVariable>
</argument>
<argument>
  <name>DNSServer</name>
  <direction>in</direction>
  <relatedStateVariable>A_ARG_TYPE_Host</relatedStateVariable>
</argument>
<argument>
  <name>NumberOfRepetitions</name>
  <direction>in</direction>
  <relatedStateVariable>A_ARG_TYPE_UInt</relatedStateVariable>
</argument>
<argument>
  <name>Timeout</name>
  <direction>in</direction>
  <relatedStateVariable>A_ARG_TYPE_MSecs</relatedStateVariable>
</argument>
<argument>
  <name>TestID</name>
  <direction>out</direction>
</argument>
<relatedStateVariable>A_ARG_TYPE_TestID</relatedStateVariable>
</argument>
</argumentList>
</action>

<action>
<name>GetNSLookupResult</name>
<argumentList>
<argument>
<name>TestID</name>
<direction>in</direction>
<relatedStateVariable>A_ARG_TYPE_TestID</relatedStateVariable>
</argument>
<argument>
<name>Status</name>
<direction>out</direction>
<relatedStateVariable>A_ARG_TYPE_NSLookupStatus</relatedStateVariable>
</argument>
<argument>
<name>AdditionalInfo</name>
<direction>out</direction>
<relatedStateVariable>A_ARG_TYPE_String</relatedStateVariable>
</argument>
<argument>
<name>SuccessCount</name>
<direction>out</direction>
<relatedStateVariable>A_ARG_TYPE_UInt</relatedStateVariable>
</argument>
<argument>
<name>Result</name>
<direction>out</direction>
<relatedStateVariable>A_ARG_TYPE_NSLookupResult</relatedStateVariable>
</argument>
</argumentList>
</action>

<action>
<name>Traceroute</name>
<argumentList>
<argument>
<name>Host</name>
<direction>in</direction>
<relatedStateVariable>A_ARG_TYPE_Host</relatedStateVariable>
</argument>
<argument>
<name>Timeout</name>
<direction>in</direction>
<relatedStateVariable>A_ARG_TYPE_MSecs</relatedStateVariable>
</argument>
<argument>
<name>DataBlockSize</name>
<direction>in</direction>
<relatedStateVariable>A_ARG_TYPE_UShort</relatedStateVariable>
</argument>
</argumentList>
</action>
<argument>
  <name>MaxHopCount</name>
  <direction>in</direction>
  <relatedStateVariable>A_ARG_TYPE_UInt</relatedStateVariable>
</argument>
<argument>
  <name>DSCP</name>
  <direction>in</direction>
  <relatedStateVariable>A_ARG_TYPE_DSCP</relatedStateVariable>
</argument>
<argument>
  <name>TestID</name>
  <direction>out</direction>
  <relatedStateVariable>A_ARG_TYPE_TestID</relatedStateVariable>
</argument>
<argument>
  <name>Status</name>
  <direction>out</direction>
  <relatedStateVariable>A_ARG_TYPE_TracerouteStatus</relatedStateVariable>
</argument>
<argument>
  <name>AdditionalInfo</name>
  <direction>out</direction>
  <relatedStateVariable>A_ARG_TYPE_String</relatedStateVariable>
</argument>
<argument>
  <name>ResponseTime</name>
  <direction>out</direction>
  <relatedStateVariable>A_ARG_TYPE_MSecs</relatedStateVariable>
</argument>
<argument>
  <name>HopHosts</name>
  <direction>out</direction>
  <relatedStateVariable>A_ARG_TYPE_Hosts</relatedStateVariable>
</argument>
<argument>
  <name>GetTracerouteResult</name>
  <argumentList>
    <argument>
      <name>TestID</name>
      <direction>in</direction>
    </argument>
    <argument>
      <name>Status</name>
      <direction>out</direction>
    </argument>
    <argument>
      <name>AdditionalInfo</name>
      <direction>out</direction>
    </argument>
    <argument>
      <name>ResponseTime</name>
      <direction>out</direction>
    </argument>
    <argument>
      <name>HopHosts</name>
      <direction>out</direction>
    </argument>
  </argumentList>
</argument>
<argument>
  <name>GetBandwidthTestInfo</name>
  <argumentList>
    <argument>
<name>BandwidthTestInfo</name>
<direction>out</direction>

<relatedStateVariable>
<name>A_ARG_TYPE_BandwidthTestInfo</name>
</relatedStateVariable>

<name>BandwidthTest</name>
<argumentList>
<argument>
<name>BandwidthTestSpec</name>
<direction>in</direction>
<relatedStateVariable>
<name>A_ARG_TYPE_BandwidthTestSpec</name>
</relatedStateVariable>
</argument>
<argument>
<name>TestEndpoint</name>
<direction>in</direction>
<relatedStateVariable>
<name>A_ARG_TYPE_TestEndpoint</name>
</relatedStateVariable>
</argument>
<argument>
<name>TestSchedule</name>
<direction>in</direction>
<relatedStateVariable>
<name>A_ARG_TYPE_TestSchedule</name>
</relatedStateVariable>
</argument>
<argument>
<name>TestSessID</name>
<direction>in</direction>
<relatedStateVariable>
<name>A_ARG_TYPE_TestSessID</name>
</relatedStateVariable>
</argument>
<argument>
<name>TestID</name>
<direction>out</direction>
<relatedStateVariable>
<name>A_ARG_TYPE_TestID</name>
</relatedStateVariable>
</argument>
</argumentList>

<name>GetBandwidthTestResult</name>
<argumentList>
<argument>
<name>TestID</name>
<direction>in</direction>
<relatedStateVariable>
<name>A_ARG_TYPE_TestID</name>
</relatedStateVariable>
</argument>
<argument>
<name>State</name>
<direction>out</direction>
<relatedStateVariable>
<name>A_ARG_TYPE_TestID</name>
</relatedStateVariable>
</argument>
</argumentList>
<relatedStateVariable> <ARG_TYPE_TestState></relatedStateVariable>

<argument>
  <name>Status</name>
  <direction>out</direction>
</argument>

<relatedStateVariable> <ARG_TYPE_BandwidthTestStatus></relatedStateVariable>

<argument>
  <name>AdditionalInfo</name>
  <direction>out</direction>
</argument>

<relatedStateVariable> <ARG_TYPE_BandwidthTestResult></relatedStateVariable>

<argument>
  <name>Result</name>
  <direction>out</direction>
</argument>

<relatedStateVariable> <ARG_TYPE_Interfaces></relatedStateVariable>

<argument>
  <name>Interfaces</name>
  <direction>in</direction>
</argument>

<relatedStateVariable> <ARG_TYPE_TestID></relatedStateVariable>

<argument>
  <name>TestID</name>
  <direction>out</direction>
</argument>

<relatedStateVariable> <ARG_TYPE_InterfaceResetStatus></relatedStateVariable>

<argument>
  <name>Status</name>
  <direction>out</direction>
</argument>
<argument>
  <name>AdditionalInfo</name>
  <direction>out</direction>
</argument>

<relatedStateVariable>A_ARG_TYPE_String</relatedStateVariable>
</argument>
<argument>
  <name>NumberOfSuccesses</name>
  <direction>out</direction>
</argument>

<relatedStateVariable>A_ARG_TYPE_UShort</relatedStateVariable>
</argument>
<argument>
  <name>NumberOfFailures</name>
  <direction>out</direction>
</argument>

<relatedStateVariable>A_ARG_TYPE_UShort</relatedStateVariable>
</argument>
</argumentList>
</action>

<action>
  <name>SelfTest</name>
  <argumentList>
    <argument>
      <name>TestID</name>
      <direction>out</direction>
    </argument>
  </argumentList>
</action>

<relatedStateVariable>A_ARG_TYPE_TestID</relatedStateVariable>
</argument>
</argumentList>
</action>

<action>
  <name>GetSelfTestResult</name>
  <argumentList>
    <argument>
      <name>TestID</name>
      <direction>in</direction>
    </argument>
    <argument>
      <name>Status</name>
      <direction>out</direction>
    </argument>
    <argument>
      <name>AdditionalInfo</name>
      <direction>out</direction>
    </argument>
  </argumentList>
</action>

<relatedStateVariable>A_ARG_TYPE_TestID</relatedStateVariable>
</argument>
</argumentList>
</action>

<action>
  <name>GetTestIDs</name>
  <argumentList>
    <argument>
      <name>Status</name>
      <direction>out</direction>
    </argument>
    <argument>
      <name>AdditionalInfo</name>
      <direction>out</direction>
    </argument>
  </argumentList>
</action>

<relatedStateVariable>A_ARG_TYPE_String</relatedVariable>
</argument>
</argumentList>
</action>

<action>
  <name>GetTestIDs</name>
</action>
<argumentList>
  <argument>
    <name>TestIDs</name>
    <direction>out</direction>
    <relatedStateVariable>TestIDs</relatedStateVariable>
  </argument>
</argumentList>
</action>

<action>
  <name>GetActiveTestIDs</name>
  <argumentList>
    <argument>
      <name>TestIDs</name>
      <direction>out</direction>
      <relatedStateVariable>ActiveTestIDs</relatedStateVariable>
    </argument>
  </argumentList>
</action>

<action>
  <name>GetTestInfo</name>
  <argumentList>
    <argument>
      <name>TestID</name>
      <direction>in</direction>
      <relatedStateVariable>A_ARG_TYPE_TestID</relatedStateVariable>
    </argument>
    <argument>
      <name>Type</name>
      <direction>out</direction>
      <relatedStateVariable>A_ARG_TYPE_TestType</relatedStateVariable>
    </argument>
    <argument>
      <name>State</name>
      <direction>out</direction>
      <relatedStateVariable>A_ARG_TYPE_TestState</relatedStateVariable>
    </argument>
  </argumentList>
</action>

<action>
  <name>CancelTest</name>
  <argumentList>
    <argument>
      <name>TestID</name>
      <direction>in</direction>
      <relatedStateVariable>A_ARG_TYPE_TestID</relatedStateVariable>
    </argument>
  </argumentList>
</action>

<action>
  <name>GetLogURIs</name>
  <argumentList>
    <argument>
      <name>LogURIs</name>
      <direction>out</direction>
      <relatedStateVariable>A_ARG_TYPE_LogURIs</relatedStateVariable>
    </argument>
  </argumentList>
</action>
<direction>out</direction>
<relatedStateVariable>LogURIs</relatedStateVariable>
</argument>
</argumentList>
</action>
</action>
<name>SetLogInfo</name>
<argumentList>
<argument>
<name>LogURI</name>
<direction>in</direction>
</argument>
<argument>
<name>Enabled</name>
<direction>in</direction>
</argument>
<argument>
<name>LogLevel</name>
<direction>in</direction>
</argument>
</argumentList>
</action>
</action>
<name>GetLogInfo</name>
<argumentList>
<argument>
<name>LogURI</name>
<direction>in</direction>
</argument>
<argument>
<name>Configurable</name>
<direction>out</direction>
</argument>
<argument>
<name>Enabled</name>
<direction>out</direction>
</argument>
<argument>
<name>LogLevel</name>
<direction>out</direction>
</argument>
</argumentList>
</action>
</action>
<name>GetLogInfo</name>
<argumentList>
<argument>
<name>LogURI</name>
<direction>in</direction>
</argument>
<argument>
<name>Configurable</name>
<direction>out</direction>
</argument>
<argument>
<name>Enabled</name>
<direction>out</direction>
</argument>
<argument>
<name>LogLevel</name>
<direction>out</direction>
</argument>
</argumentList>
</action>
<direction>out</direction>

<relatedStateVariable>A_ARG_TYPE_LogURL</relatedStateVariable>
</argument>
<argument>
  <name>MaxSize</name>
  <direction>out</direction>
</argument>

<relatedStateVariable>A_ARG_TYPE_LogMaxSize</relatedStateVariable>
</argument>
<argument>
  <name>LastChange</name>
  <direction>out</direction>
</argument>

<relatedStateVariable>A_ARG_TYPE_DateTime</relatedStateVariable>
</argument>
</argumentList>
</action>

<action>
  <name>GetACLData</name>
  <argumentList>
    <argument>
      <name>ACL</name>
      <direction>out</direction>
      <relatedStateVariable>A_ARG_TYPE_ACL</relatedStateVariable>
    </argument>
  </argumentList>
</action>

<serviceStateTable>
  <stateVariable sendEvents="yes">
    <name>DeviceStatus</name>
    <dataType>string</dataType>
  </stateVariable>
  <stateVariable sendEvents="yes">
    <name>SequenceMode</name>
    <dataType>boolean</dataType>
    <defaultValue>0</defaultValue>
  </stateVariable>
  <stateVariable sendEvents="yes">
    <name>TestIDs</name>
    <dataType>string</dataType>
    <defaultValue></defaultValue>
  </stateVariable>
  <stateVariable sendEvents="yes">
    <name>ActiveTestIDs</name>
    <dataType>string</dataType>
    <defaultValue></defaultValue>
  </stateVariable>
  <stateVariable sendEvents="yes">
    <name>LogURIs</name>
    <dataType>string</dataType>
  </stateVariable>
  <stateVariable sendEvents="no">
    <name>A_ARG_TYPE_Boolean</name>
    <dataType>boolean</dataType>
  </stateVariable>
</serviceStateTable>
<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_String</name>
  <dataType>string</dataType>
</stateVariable>

<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_UShort</name>
  <dataType>ui2</dataType>
</stateVariable>

<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_UInt</name>
  <dataType>ui4</dataType>
</stateVariable>

<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_DateTime</name>
  <dataType>dateTime.tz</dataType>
</stateVariable>

<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_MSecs</name>
  <dataType>ui4</dataType>
</stateVariable>

<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_RebootStatus</name>
  <dataType>string</dataType>
  <allowedValueList>
    <allowedValue>RebootNow</allowedValue>
    <allowedValue>RebootLater</allowedValue>
  </allowedValueList>
</stateVariable>

<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_TestID</name>
  <dataType>ui4</dataType>
</stateVariable>

<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_TestType</name>
  <dataType>string</dataType>
  <allowedValueList>
    <allowedValue>BandwidthTest</allowedValue>
    <allowedValue>InterfaceReset</allowedValue>
    <allowedValue>NSLookup</allowedValue>
    <allowedValue>Ping</allowedValue>
    <allowedValue>SelfTest</allowedValue>
    <allowedValue>Traceroute</allowedValue>
  </allowedValueList>
</stateVariable>

<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_TestState</name>
  <dataType>string</dataType>
  <allowedValueList>
    <allowedValue>Requested</allowedValue>
    <allowedValue>InProgress</allowedValue>
    <allowedValue>Canceled</allowedValue>
    <allowedValue>Completed</allowedValue>
  </allowedValueList>
</stateVariable>

<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_TestEndpoint</name>
  <dataType>string</dataType>
</stateVariable>
<allowedValueList>
    <allowedValue>Server</allowedValue>
    <allowedValue>Client</allowedValue>
</allowedValueList>
</stateVariable>
<stateVariable sendEvents="no">
    <name>A_ARG_TYPE_TestSchedule</name>
    <dataType>string</dataType>
</stateVariable>
<stateVariable sendEvents="no">
    <name>A_ARG_TYPE_TestSessID</name>
    <dataType>string</dataType>
</stateVariable>
<stateVariable sendEvents="no">
    <name>A_ARG_TYPE_DSCP</name>
    <dataType>ui1</dataType>
    <allowedValueRange>
        <minimum>0</minimum>
        <maximum>63</maximum>
    </allowedValueRange>
</stateVariable>
<stateVariable sendEvents="no">
    <name>A_ARG_TYPE_Host</name>
    <dataType>string</dataType>
</stateVariable>
<stateVariable sendEvents="no">
    <name>A_ARG_TYPE_Hosts</name>
    <dataType>string</dataType>
</stateVariable>
<stateVariable sendEvents="no">
    <name>A_ARG_TYPE_HostName</name>
    <dataType>string</dataType>
</stateVariable>
<stateVariable sendEvents="no">
    <name>A_ARG_TYPE_PingStatus</name>
    <dataType>string</dataType>
    <allowedValueList>
        <allowedValue>Success</allowedValue>
        <allowedValue>Error_CannotResolveHostName</allowedValue>
        <allowedValue>Error_Internal</allowedValue>
        <allowedValue>Error_Other</allowedValue>
    </allowedValueList>
</stateVariable>
<stateVariable sendEvents="no">
    <name>A_ARG_TYPE_NSLookupStatus</name>
    <dataType>string</dataType>
    <allowedValueList>
        <allowedValue>Success</allowedValue>
        <allowedValue>Error_DNSServerNotResolved</allowedValue>
        <allowedValue>Error_Internal</allowedValue>
        <allowedValue>Error_Other</allowedValue>
    </allowedValueList>
</stateVariable>
<stateVariable sendEvents="no">
    <name>A_ARG_TYPE_NSLookupResult</name>
    <dataType>string</dataType>
</stateVariable>
<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_TracerouteStatus</name>
  <dataType>string</dataType>
  <allowedValueList>
    <allowedValue>Success</allowedValue>
    <allowedValue>Error_CannotResolveHostName</allowedValue>
    <allowedValue>Error_MaxHopCountExceeded</allowedValue>
    <allowedValue>Error_Internal</allowedValue>
    <allowedValue>Error_Other</allowedValue>
  </allowedValueList>
</stateVariable>

<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_BandwidthTestInfo</name>
  <dataType>string</dataType>
</stateVariable>

<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_BandwidthTestSpec</name>
  <dataType>string</dataType>
</stateVariable>

<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_BandwidthTestStatus</name>
  <dataType>string</dataType>
  <allowedValueList>
    <allowedValue>Success</allowedValue>
    <allowedValue>Error_InitConnectionFailed</allowedValue>
    <allowedValue>Error_NoResponse</allowedValue>
    <allowedValue>Error_TransferFailed</allowedValue>
    <allowedValue>Error_PasswordRequestFailed</allowedValue>
    <allowedValue>Error_LoginFailed</allowedValue>
    <allowedValue>Error_NoTransferMode</allowedValue>
    <allowedValue>Error_IncorrectSize</allowedValue>
    <allowedValue>Error_NoTransferComplete</allowedValue>
    <allowedValue>Error_Timeout</allowedValue>
    <allowedValue>Error_Other</allowedValue>
    <allowedValue>Error_FTPNoPASV</allowedValue>
    <allowedValue>Error_FTPNoCWD</allowedValue>
    <allowedValue>Error_FTPNoSTOR</allowedValue>
  </allowedValueList>
</stateVariable>

<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_BandwidthTestResult</name>
  <dataType>string</dataType>
</stateVariable>

<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_Interfaces</name>
  <dataType>string</dataType>
  <allowedValueList>
    <allowedValue>AllInterfaces</allowedValue>
    <allowedValue>RequestInterface</allowedValue>
    <allowedValue>NorthboundInterfaces</allowedValue>
  </allowedValueList>
</stateVariable>

<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_InterfaceResetStatus</name>
  <dataType>string</dataType>
  <allowedValueList>
    <allowedValue>Success</allowedValue>
  </allowedValueList>
</stateVariable>
<allowedValue>Error</allowedValue>
</allowedValueList>
</stateVariable>
<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_LogURI</name>
  <dataType>uri</dataType>
</stateVariable>
<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_LogURL</name>
  <dataType>uri</dataType>
</stateVariable>
<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_LogLevel</name>
  <dataType>string</dataType>
  <allowedValueList>
    <allowedValue>Emergency</allowedValue>
    <allowedValue>Alert</allowedValue>
    <allowedValue>Critical</allowedValue>
    <allowedValue>Error</allowedValue>
    <allowedValue>Warning</allowedValue>
    <allowedValue>Notice</allowedValue>
    <allowedValue>Informational</allowedValue>
    <allowedValue>Debug</allowedValue>
  </allowedValueList>
</stateVariable>
<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_LogMaxSize</name>
  <dataType>ui4</dataType>
</stateVariable>
<stateVariable sendEvents="no">
  <name>A_ARG_TYPE_ACL</name>
  <dataType>string</dataType>
</stateVariable>
</serviceStateTable>
</scpd>
4. **XML Schema (Normative)**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:bms="urn:schemas-upnp-org:dm:bms"
    targetNamespace="urn:schemas-upnp-org:dm:bms"
    elementFormDefault="unqualified"
    attributeFormDefault="unqualified" version="2-20120216">
  <xs:simpleType name="ActionName">
    <xs:annotation>
      <xs:documentation>UPnP action name.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:token"/>
  </xs:simpleType>
  <xs:simpleType name="Direction">
    <xs:annotation>
      <xs:documentation>Bandwidth test data flow direction (as capability and in test spec).</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:token">
      <xs:enumeration value="Put"/>
      <xs:enumeration value="Get"/>
      <xs:enumeration value="PutEcho"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="FilterExpression">
    <xs:annotation>
      <xs:documentation>ACL filter expression. Syntax is based on XPath 1.0.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:simpleType name="IntervalRange">
    <xs:annotation>
      <xs:documentation>Bandwidth test interval range of form \d+:\d+.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:token">
      <xs:pattern value="\d+:\d+"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="IPAddress">
    <xs:annotation>
      <xs:documentation>IPv4/IPv6 address.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:simpleType name="ProfileName">
    <xs:restriction base="xs:token">
      <xs:enumeration value="Baseline"/>
      <xs:enumeration value="BBF"/>
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="ProfileNames">
    <xs:list itemType="bms:ProfileName"/>
  </xs:simpleType>
</xs:schema>
```
<xs:sequence>
  <xs:element name="Minimum" type="xs:double"/>
  <xs:element name="Maximum" type="xs:double"/>
  <xs:element minOccurs="0" name="Step" type="xs:double"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:complexType>
<xs:element minOccurs="0" name="AllowedValueList">
  <xs:complexType>
    <xs:sequence maxOccurs="unbounded">
      <xs:element name="AllowedValue" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:complexType>
<xs:complexType name="StringCapability">
  <xs:annotation>
    <xs:documentation>Bandwidth test string capability.</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="bms:Capability">
      <xs:sequence minOccurs="0">
        <xs:element minOccurs="0" name="AllowedValueList">
          <xs:complexType>
            <xs:sequence maxOccurs="unbounded">
              <xs:element name="AllowedValue" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:complexType>
<xs:attributeGroup name="ProtocolKey">
  <xs:annotation>
    <xs:documentation>Bandwidth test protocol unique key attributes.</xs:documentation>
  </xs:annotation>
  <xs:attribute name="Name" use="required" type="bms:ProtocolName"/>
  <xs:attribute default="" name="Version" type="xs:token"/>
  <xs:attribute name="Profile" use="required" type="bms:ProfileName"/>
</xs:attributeGroup>
<xs:attributeGroup name="ResultKey">
  <xs:annotation>
    <xs:documentation>Bandwidth test result unique key attributes.</xs:documentation>
  </xs:annotation>
  <xs:attribute name="Interval" type="bms:IntervalRange"/>
  <xs:attribute name="Stream" type="xs:unsignedInt"/>
</xs:attributeGroup>
<xs:group name="SettingsCaps">
<xs:annotation>
  <xs:documentation>Bandwidth test settings capabilities.</xs:documentation>
</xs:annotation>
<xs:sequence>
  <xs:element minOccurs="0" name="Interface" type="bms:StringCapability">
    <xs:annotation>
      <xs:documentation>Supported IP interface names</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element minOccurs="0" name="Transport" type="bms:StringCapability">
    <xs:annotation>
      <xs:documentation>TCP, UDP, ...</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element minOccurs="0" name="Host" type="bms:StringCapability">
    <xs:annotation>
      <xs:documentation>Supported hosts (more likely, no list, so just indicates that Host is supported)</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element minOccurs="0" name="Port" type="bms:NumericCapability">
    <xs:annotation>
      <xs:documentation>[0:65535]</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element minOccurs="0" name="TLSPort" type="bms:NumericCapability">
    <xs:annotation>
      <xs:documentation>[0:65535]</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element minOccurs="0" name="SockBuffBytes" type="bms:NumericCapability">
    <xs:annotation>
      <xs:documentation>[0:]</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element minOccurs="0" name="AppBuffBytes" type="bms:NumericCapability">
    <xs:annotation>
      <xs:documentation>[0:]</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element minOccurs="0" name="Direction" type="bms:StringCapability">
    <xs:annotation>
      <xs:documentation>Push, Pull, PushPull, PullPush</xs:documentation>
    </xs:annotation>
  </xs:element>
</xs:sequence>
<xs:element name="TimeMSecs" type="bms:NumericCapability">
  <xs:annotation>
    <xs:documentation %[0:]
  </xs:annotation>
</xs:element>

<xs:element name="LengthBytes" type="bms:NumericCapability">
  <xs:annotation>
    <xs:documentation %[0:]
  </xs:annotation>
</xs:element>

<xs:element name="Username" type="bms:StringCapability">
  <xs:annotation>
    <xs:documentation>Supported usernames</xs:documentation>
  </xs:annotation>
</xs:element>

<xs:element name="Password" type="bms:StringCapability">
  <xs:annotation>
    <xs:documentation>Supported passwords (more likely, no list, so just indicates that Password is supported)</xs:documentation>
  </xs:annotation>
</xs:element>

<xs:element name="Path" type="bms:StringCapability">
  <xs:annotation>
    <xs:documentation>Supported paths (more likely, no list, so just indicates that Path is supported)</xs:documentation>
  </xs:annotation>
</xs:element>

<xs:element name="BandwidthBytesPerSec" type="bms:NumericCapability">
  <xs:annotation>
    <xs:documentation>[0:]
  </xs:annotation>
</xs:element>

<xs:element name="DSCP" type="bms:NumericCapability">
  <xs:annotation>
    <xs:documentation>[0:63]
  </xs:annotation>
</xs:element>

<xs:element name="EthernetPriority" type="bms:NumericCapability">
  <xs:annotation>
    <xs:documentation>[0:7]
  </xs:annotation>
</xs:element>

<xs:element name="IntervalMSecs" type="bms:NumericCapability">
  <xs:annotation>
    <xs:documentation>[0:]
  </xs:annotation>
</xs:element>

Copyright UPnP Forum © 2012. All rights reserved.
<xs:element minOccurs="0" name="Threads" type="bms:NumericCapability"/>
<xs:element minOccurs="0" name="TCPNoDelay" type="bms:BooleanCapability"/>
<xs:element minOccurs="0" name="TCPMSS" type="bms:NumericCapability"/>
<xs:element minOccurs="0" name="ExitWhenDone" type="bms:BooleanCapability"/>

<xs:group name="ResultsCaps">
  <xs:element minOccurs="0" name="TCPSYNTime" type="bms:Empty"/>
  <xs:element minOccurs="0" name="TCPSYNACKTime" type="bms:Empty"/>
  <xs:element minOccurs="0" name="TCPLateACKs" type="bms:Empty"/>
  <xs:element minOccurs="0" name="ROMTime" type="bms:Empty"/>
  <xs:element minOccurs="0" name="BOMTime" type="bms:Empty"/>
  <xs:element minOccurs="0" name="EOMTime" type="bms:Empty"/>
  <xs:element minOccurs="0" name="TestBytesSent" type="bms:Empty"/>
  <xs:element minOccurs="0" name="TestBytesReceived" type="bms:Empty"/>
  <xs:element minOccurs="0" name="TotalBytesSent" type="bms:Empty"/>
  <xs:element minOccurs="0" name="TotalBytesReceived" type="bms:Empty"/>
  <xs:element minOccurs="0" name="TestPacketsSent" type="bms:Empty"/>
  <xs:element minOccurs="0" name="TestPacketsReceived" type="bms:Empty"/>
  <xs:element minOccurs="0" name="TotalPacketsSent" type="bms:Empty"/>
  <xs:element minOccurs="0" name="TotalPacketsReceived" type="bms:Empty"/>
  <xs:element minOccurs="0" name="DuplicatePackets" type="bms:Empty"/>
  <xs:element minOccurs="0" name="LostPackets" type="bms:Empty"/>
</xs:group>
<xs:element minOccurs="0" name="OutOfSequencePackets" type="bms:Empty"/>
<xs:element minOccurs="0" name="ResentPackets" type="bms:Empty"/>
<xs:element minOccurs="0" name="LatencyUSecs" type="bms:Empty"/>
<xs:element minOccurs="0" name="JitterUSecs" type="bms:Empty"/>
</xs:sequence>
</xs:group>
<xs:group name="FilesCaps">
<xs:annotation>
<xs:documentation>Bandwidth test files capabilities.</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element maxOccurs="unbounded" minOccurs="0" name="File">
<xs:complexType>
<xs:sequence>
<xs:element name="Path" type="xs:string"/>
<xs:element default="1" minOccurs="0" name="RealFile" type="xs:boolean"/>
<xs:element minOccurs="0" name="SizeBytes" type="xs:unsignedLong"/>
<xs:element minOccurs="0" name="MIMEType" type="xs:token"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:group>
<xs:group name="Settings">
<xs:annotation>
<xs:documentation>Bandwidth test settings.</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element minOccurs="0" name="Interface" type="xs:token"/>
<xs:element minOccurs="0" name="Transport">
<xs:simpleType>
<xs:restriction base="xs:token">
<xs:enumeration value="TCP"/>
<xs:enumeration value="UDP"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element minOccurs="0" name="Host" type="xs:token"/>
<xs:element minOccurs="0" name="Port" type="xs:unsignedShort"/>
<xs:element minOccurs="0" name="TLSPort" type="xs:unsignedShort"/>
<xs:element minOccurs="0" name="SockBuffBytes" type="xs:unsignedInt"/>
<xs:element minOccurs="0" name="AppBuffBytes" type="xs:unsignedInt"/>
<xs:element minOccurs="0" name="Direction" type="bms:Direction"/>
<xs:element minOccurs="0" name="TimeMSecs" type="xs:unsignedInt"/>
<xs:element minOccurs="0" name="LengthBytes" type="xs:unsignedLong"/>
<xs:element minOccurs="0" name="Username" type="xs:string"/>
<xs:element minOccurs="0" name="Password" type="xs:string"/>
</xs:sequence>
</xs:group>
<xs:documentation>SHOULD always read back as empty string.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element minOccurs="0" name="Path" type="xs:string"/>
<xs:element minOccurs="0" name="BandwidthBytesPerSec" type="xs:unsignedInt"/>
<xs:element default="0" minOccurs="0" name="DSCP">
<xs:simpleType>
<xs:restriction base="xs:unsignedInt">
<xs:minInclusive value="0"/>
<xs:maxInclusive value="63"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element default="0" minOccurs="0" name="EthernetPriority">
<xs:simpleType>
<xs:restriction base="xs:unsignedInt">
<xs:minInclusive value="0"/>
<xs:maxInclusive value="7"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element minOccurs="0" name="IntervalMSecs" type="xs:unsignedInt"/>
<xs:element minOccurs="0" name="Threads" type="xs:unsignedInt"/>
<xs:element default="0" minOccurs="0" name="TCPNoDelay" type="xs:boolean"/>
<xs:element minOccurs="0" name="TCPMSS" type="xs:unsignedInt"/>
<xs:element default="0" minOccurs="0" name="ExitWhenDone" type="xs:boolean"/>
</xs:sequence>
</xs:group>
<xs:group name="SpecSettings">
<xs:annotation>
<xs:documentation>Bandwidth test spec settings (top-level settings plus optional endpoint-specific settings).</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:group ref="bms:Settings"/>
<xs:element minOccurs="0" name="Server">
<xs:complexType>
<xs:group ref="bms:Settings"/>
</xs:complexType>
</xs:element>
<xs:element minOccurs="0" name="Client">
<xs:complexType>
<xs:group ref="bms:Settings"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:group>
<xs:group name="EndpointInfo">
<xs:annotation>
<xs:documentation>Bandwidth test endpoint (server or client) capabilities and active instances.</xs:documentation>
</xs:annotation>

<xs:sequence>
  <xs:group ref="bms:SettingsCaps"/>
  <xs:group ref="bms:FilesCaps"/>
  <xs:group ref="bms:ResultsCaps"/>
  <xs:element maxOccurs="unbounded" minOccurs="0" name="Instance">XXX Was SpecSettings, but Settings is correct?"</xs:element>
</xs:sequence>
</xs:group>

<xs:group name="Results">
  <xs:annotation>
    <xs:documentation>Bandwidth test results.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element minOccurs="0" name="TCPSYNTime" type="xs:dateTime"/>
    <xs:element minOccurs="0" name="TCPSYNACKTime" type="xs:dateTime"/>
    <xs:element minOccurs="0" name="TCPLateACKs" type="xs:unsignedInt"/>
    <xs:element minOccurs="0" name="ROMTime" type="xs:dateTime"/>
    <xs:element minOccurs="0" name="BOMTime" type="xs:dateTime"/>
    <xs:element minOccurs="0" name="EOMTime" type="xs:dateTime"/>
    <xs:element minOccurs="0" name="TestBytesSent" type="xs:unsignedLong"/>
    <xs:element minOccurs="0" name="TestBytesReceived" type="xs:unsignedLong"/>
    <xs:element minOccurs="0" name="TotalBytesSent" type="xs:unsignedLong"/>
    <xs:element minOccurs="0" name="TotalBytesReceived" type="xs:unsignedLong"/>
    <xs:element minOccurs="0" name="TestPacketsSent" type="xs:unsignedInt"/>
    <xs:element minOccurs="0" name="TestPacketsReceived" type="xs:unsignedInt"/>
    <xs:element minOccurs="0" name="TotalPacketsSent" type="xs:unsignedInt"/>
    <xs:element minOccurs="0" name="TotalPacketsReceived" type="xs:unsignedInt"/>
    <xs:element minOccurs="0" name="DuplicatePackets" type="xs:unsignedInt"/>
    <xs:element minOccurs="0" name="LostPackets" type="xs:unsignedInt"/>
    <xs:element minOccurs="0" name="OutOfSequencePackets" type="xs:unsignedInt"/>
    <xs:element minOccurs="0" name="ResentPackets" type="xs:unsignedInt"/>
    <xs:element minOccurs="0" name="LatencyUSecs" type="xs:int"/>
    <xs:element minOccurs="0" name="JitterUSecs" type="xs:int"/>
  </xs:sequence>
</xs:group>
<xs:complexType name="NSLookupResult">
  <xs:annotation>
    <xs:documentation>NSLookup result (content of NSLookupResult element).</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="Result" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>Results from the most recent invocation of the test, one instance per repetition.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Status">
            <xs:annotation>
              <xs:documentation>Result Parameter to represent whether the NS Lookup was successful or not.</xs:documentation>
            </xs:annotation>
            <xs:simpleType>
              <xs:restriction base="xs:token">
                <xs:enumeration value="Success"/>
                <xs:enumeration value="Error_DNSServerNotAvailable"/>
                <xs:enumeration value="Error_HostNameNotResolved"/>
                <xs:enumeration value="Error_Timeout"/>
                <xs:enumeration value="Error_Other"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
          <xs:element name="AnswerType">
            <xs:annotation>
              <xs:documentation>Result parameter to represent whether the answer is Authoritative or not.</xs:documentation>
            </xs:annotation>
            <xs:simpleType>
              <xs:restriction base="xs:token">
                <xs:enumeration value="None"/>
                <xs:enumeration value="Authoritative"/>
                <xs:enumeration value="NonAuthoritative"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
          <xs:element name="HostNameReturned">
            <xs:annotation>
              <xs:documentation>Result parameter to represent the fully qualified name for the Host Name in the calling parameter (e.g. HostName.DomainName); if no response was provided, then this parameter is an empty string.</xs:documentation>
            </xs:annotation>
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:maxLength value="256"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
          <xs:element name="IPAddresses"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:annotation>
  <xs:documentation>
  Result parameter to represent the list of one or more comma-separated IPv4/IPv6 addresses returned by the NS Lookup; if no response was provided, then this parameter is an empty string.
  </xs:documentation>
</xs:annotation>
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:maxLength value="256"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name="DNSServerIP" type="bms:IPAddress">
  <xs:annotation>
    <xs:documentation>
    Result parameter to represent the actual DNS Server IPv4/IPv6 address that the NS Lookup used.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="ResponseTime" type="xs:unsignedInt">
  <xs:annotation>
    <xs:documentation>
    Response time (for the first response packet) in milliseconds, or 0 if no response was received.
    </xs:documentation>
  </xs:annotation>
</xs:element>
</xs:complexType>
<xs:complexType name="BandwidthTestInfo">
  <xs:annotation>
    <xs:documentation>
    Bandwidth test info (content of BandwidthTestInfo element).
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element maxOccurs="unbounded" name="Protocol">
      <xs:complexType>
        <xs:sequence>
          <xs:element minOccurs="0" name="Server">
            <xs:complexType>
              <xs:group ref="bms:EndpointInfo"/>
            </xs:complexType>
          </xs:element>
          <xs:element minOccurs="0" name="Client">
            <xs:complexType>
              <xs:group ref="bms:EndpointInfo"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        <xs:attributeGroup ref="bms:ProtocolKey"/>
        <xs:attribute default="" name="BaseProfiles" type="bms:ProfileNames"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="BandwidthTestSpec">
    <xs:annotation>
        <xs:documentation>Bandwidth test spec (content of BandwidthTestSpec element).</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="Protocol">
            <xs:complexType>
                <xs:group ref="bms:SpecSettings"/>
                <xs:attributeGroup ref="bms:ProtocolKey"/>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="BandwidthTestResult">
    <xs:annotation>
        <xs:documentation>Bandwidth test result (content of BandwidthTestResult element).</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="Protocol">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="Spec">
                        <xs:complexType>
                            <xs:group ref="bms:SpecSettings"/>
                        </xs:complexType>
                    </xs:element>
                    <xs:choice maxOccurs="unbounded" minOccurs="0">
                        <xs:element name="Server">
                            <xs:complexType>
                                <xs:group ref="bms:Results"/>
                                <xs:attributeGroup ref="bms:ResultKey"/>
                            </xs:complexType>
                        </xs:element>
                        <xs:element name="Client">
                            <xs:complexType>
                                <xs:group ref="bms:Results"/>
                                <xs:attributeGroup ref="bms:ResultKey"/>
                            </xs:complexType>
                        </xs:element>
                    </xs:choice>
                    <xs:attributeGroup ref="bms:ProtocolKey"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="ACLEntry">
    <xs:annotation>
        <xs:documentation>Individual ACL entry.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="Action" type="bms:ActionName">
            <xs:annotation>
                <xs:documentation>The name of the action to which this ACL entry refers.</xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<xs:annotation>
</xs:element>
<xs:element name="Filter" type="bms:FilterExpression">
  <xs:documentation>ACL filter expression that determines whether this entry matches a given action invocation.</xs:documentation>
</xs:element>
<xs:element name="Roles">
  <xs:documentation>List of Roles that are permitted to invoke this action if the filter matches.</xs:documentation>
</xs:element>
<xs:element name="ACL">
  <xs:documentation>ACL data (content of ACL element).</xs:documentation>
</xs:element>
<xs:element name="NSLookupResult" type="bms:NSLookupResult">
  <xs:documentation>GetNSLookupResult() result.</xs:documentation>
</xs:element>
<xs:element name="BandwidthTestInfo" type="bms:BandwidthTestInfo">
  <xs:documentation>GetBandwidthTestInfo() result.</xs:documentation>
</xs:element>
<xs:element name="BandwidthTestSpec" type="bms:BandwidthTestSpec">
  <xs:documentation>BandwidthTest() argument.</xs:documentation>
</xs:element>
<xs:element name="BandwidthTestResult" type="bms:BandwidthTestResult">
  <xs:documentation>GetBandwidthTestResult() result.</xs:documentation>
</xs:element>
<xs:element name="ACL">
  <xs:documentation>GetACLData() result.</xs:documentation>
</xs:element>
<xs:unique name="ACLEntry">
  <xs:selector xpath="ACLEntry"/>
  <xs:field xpath="Action"/>
  <xs:field xpath="Filter"/>
</xs:unique>
</xs:element>

<xs:element name="Test">
  <xs:annotation>
    <xs:documentation>This element is not used by any actions but is useful for test data because a single document can contain any mixture of top-level elements.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:choice maxOccurs="unbounded" minOccurs="0">
      <xs:element name="NSLookupResult" type="bms:NSLookupResult"/>
      <xs:element name="BandwidthTestInfo" type="bms:BandwidthTestInfo"/>
      <xs:element name="BandwidthTestSpec" type="bms:BandwidthTestSpec"/>
      <xs:element name="BandwidthTestResult" type="bms:BandwidthTestResult"/>
      <xs:element name="ACL" type="bms:ACL"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
5. Version History (Informative)

**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.