BasicManagement:1
Service Template Version 1.01

For UPnP Version 1.0
Status: Standardized DCP (SDCP)
Date: July 20, 2010

Authors	Company
William Lupton (Editor)	2Wire
Francois-Gaël Ottogalli	France Telecom Group
Kiran Vedula	Samsung Electronics
Davide Moreo	Telecom Italia

Contents

List of Tables

List of Figures

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:1* 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 or self test.
- Enabling / disabling logging and retrieving log files.

Most of the service's features are optional: only the *DeviceStatus* state variable and the *GetDeviceStatus()* action are mandatory. This means that only a very small amount of effort is necessary in order to add basic management capability to a UPnP device.

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:1* 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:1* 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:

[CMS] *UPnP ConfigurationManagement:1 Service Document*, UPnP Forum, July 20, 2010, http://www.upnp.org/specs/dm/UPnP-dm-ConfigurationManagement-v1.0-Service.pdf.

[CMS-XSD] *XML Schema for ConfigurationManagement:1*, UPnP Forum, July 5, 2010, http://www.upnp.org/schemas/dm/cms-v1.xsd.

[DEVICE] *UPnP ManageableDevice:1 Device Document*, UPnP Forum, July 20, 2010, http://www.upnp.org/specs/dm/UPnP-dm-ManageableDevice-v1.0-Device.pdf.

[DNS]	*RFC 1035, Domain Names - Implementation and Specification*, IETF, November 1987, http://tools.ietf.org/html/rfc1035
[DSCP]	*RFC 2474, Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*, IETF, December 1988, http://tools.ietf.org/html/rfc2474
[HTTP]	*RFC 2616, Hypertext Transfer Protocol – HTTP/1.1*, IETF, June 1999, http://tools.ietf.org/html/rfc2616
[ICMP]	*RFC 792, Internet Control Message Protocol*, IETF, September 1981, http://tools.ietf.org/html/rfc792
[REQLEV]	*RFC 2119, Key words for use in RFCs to Indicate Requirement Levels*, IETF, March 1997, http://tools.ietf.org/html/rfc2119
[SYSLOG]	*RFC 3164, The BSD syslog Protocol*, IETF, August 2001, http://tools.ietf.org/html/rfc3164
[UDA1.0]	UPnP Device Architecture version 1.0, UPnP Forum, October 2008, http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0.pdf
[URI]	*RFC 3986, Uniform Resource Identifier (URI): Generic Syntax*, IETF, January 2005, http://tools.ietf.org/html/rfc3986
[XML]	*Extensible Markup Language (XML) 1.0 (Third Edition)*, W3C, February 2004, http://www.w3.org/TR/2004/REC-xml-20040204
[XML-NS]	*The "xml:" Namespace*, W3C, April 2006, http://www.w3.org/XML/1998/namespace
[XML-NMSP]	*Namespaces in XML*, W3C, August 2006, http://www.w3.org/TR/REC-xml-names
[XML-SCHEMA-1]	*XML Schema Part 1: Structures Second Edition*, W3C, October 2004, http://www.w3.org/TR/xmlschema-1
[XML-SCHEMA-2]	*XML Schema Part 2: Datatypes Second Edition*, W3C, October 2004, http://www.w3.org/TR/xmlschema-2

1.3. Glossary

BMS	*BasicManagement* Service
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.

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

Type refinement of string	Value	Comments
CSV (**string**)	"first,second"	List of 2 strings used as state variable or action argument value.
CSV (`xsd:string`)	"first,second"	List of 2 strings used within an XML element
CSV (`xsd:token`)	"first, second "	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.
CSV (**string**, **dateTime.tz** [, **string**])	"Warning,2009-07-07T13:22:41, third\,value"	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".
CSV (**string**, **dateTime.tz** [, **string**])	"Warning,2009-07-07T13:22:41,"	As above but third value is empty.

Type refinement of string	Value	Comments
CSV (string, date-Time.tz [, string])	“Warning,2009-07-07T13:22:41”	As above but third value is omitted.
CSV (*A_ARG_TYPE_-Host*)	“grumpy,sleepy”	List of data items used as action argument value, each of which obeys the rules governing *A_ARG_TYPE_Host*. Any comma or backslash characters within a data item would have been escaped.
CSV (i4)	“1, 2”	Illegal CSV. White space is not allowed as part of an integer value.
CSV (string)	“a,,c,”	List of 4 strings “a”, “”, “c” and “”.
CSV (string)	“”	Empty list. It is not possible to create a list containing a single empty string.

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.

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, or should be, globally unique. It has a single definition that is accessible to anyone who uses the namespace. It has the same meaning anywhere that it is used, both inside and outside XML documents. The namespace prefix, however, in formal XML usage, is defined only in an XML document. It must be locally unique to the document. Any valid XML no-colon-name may be used. And, in formal XML usage, no two XML documents are ever required to use the same namespace prefix to refer to the same namespace. The creation and use of the namespace prefix was standardized by the W3C XML Committee in [XML-NMSP] strictly as a convenient local shorthand replacement for the full URI name of a namespace in individual documents.

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

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

Table 1-2: Namespace Definitions

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

Table 1-3: Schema-related Information

Standard Name-space Prefix	Relative URI and File Name[1] • Form 1, 2, 3	Valid Root Element(s)	Schema Reference
DM Working Committee defined namespaces			

Standard Name-space Prefix	Relative URI and File Name[1] • Form 1, 2, 3	Valid Root Element(s)	Schema Reference

cms	• cms-v*n*-*yyyymmdd*.xsd
	• cms-v*n*.xsd
	• cms.xsd
		`<StructurePathList>`
		`<ParameterPathList>`
		`<ParameterAttributeList>`
		`<InstanceValueList>`
		`<SupportedDataModels>`
		`<InstancePathList>`
		`<ContentPathList>`
		`<AttributePathList>`
			[CMS]

bmsnsl	• bmsnsl-v*n*-*yyyymmdd*.xsd • bmsnsl-v*n*.xsd • bmsnsl.xsd	`<NSLookupResult>`	[BMS]

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

1.6.1. 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)

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

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

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

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

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

As an example, the original schema URI for the "`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***

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

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 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".

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:1*

2.2. Key Concepts

Basic management operations fall into three categories: maintenance, diagnostic tests, and logging.

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.

Diagnostic tests:

- 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 there is a pair of actions for each diagnostic test, one to request the test and one 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, or retain more than one set of results per test type. Test IDs MUST become invalid when the corresponding test results are discarded. Individual tests 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.

Logging:

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

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

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

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

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

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

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

2.3. State Variables

Table 2-1: State Variables

Variable Name	Req. or Opt.[1]	Data Type	Allowed Value	Default Value	Eng. Units
DeviceStatus	*R*	**string**	CSV[2] (**string**, **dateTime.tz** [, **string**]), where first string is: *OK, Warning, Error, Fatal* (section 2.3.1)		
SequenceMode	*O*	**boolean**	(section 2.3.2)	"**0**"	
ActiveTestIDs	*O*	**string**	CSV (*A_ARG-_TYPE_TestID*) (section 2.3.3)	""	
LogURIs	*O*	**string**	CSV (*A_ARG-_TYPE_Log-URI*) (section 2.3.4)		
A_ARG_TYPE_Boolean	*O*	**boolean**	(section 2.3.4)		
A_ARG_TYPE_String	*O*	**string**	(section 2.3.6)		
A_ARG_TYPE_UShort	*O*	**ui2**	(section 2.3.7)		
A_ARG_TYPE_UInt	*O*	**ui4**	(section 2.3.8)		
A_ARG_TYPE_DateTime	*O*	**dateTime.tz**	(section 2.3.9)		
A_ARG_TYPE_MSecs	*O*	**ui4**	(section 2.3.10)		Milliseconds
A_ARG_TYPE_RebootStatus	*O*	**string**	*RebootNow, RebootLater* (section 2.3.11)		
A_ARG_TYPE_TestID	*O*	**ui4**	(section 2.3.12)		
A_ARG_TYPE_TestType	*O*	**string**	*NSLookup*, *Ping*, *SelfTest*, *Traceroute* (section 2.3.13)		
A_ARG_TYPE_TestState	*O*	**string**	*Requested*, *InProgress*, *Canceled*, *Completed* (section 2.3.14)		
A_ARG_TYPE_DSCP	*O*	**ui1**	Between 0 and 63 inclusive (section 2.3.15)		

Variable Name	Req. or Opt.[1]	Data Type	Allowed Value	Default Value	Eng. Units
A_ARG_TYPE_Host	*O*	**string**	(section 2.3.16)		
A_ARG_TYPE_Hosts	*O*	**string**	CSV (*A_ARG-_TYPE_Host*) (section 2.3.17)		
A_ARG_TYPE_HostName	*O*	**string**	(section 2.3.18)		
A_ARG_TYPE_PingStatus	*O*	**string**	*Success*, *Error_Can-notResolve-HostName* (section 2.3.19)		
A_ARG_TYPE_NSLookupStatus	*O*	**string**	*Success*, *Error_DNS-ServerNot-Resolved* (section 2.3.20)		
A_ARG_TYPE_NSLookupResult	*O*	**string**	(section 2.3.21)		
A_ARG_TYPE_TracerouteStatus	*O*	**string**	*Success*, *Error_Can-notResolve-HostName*, *Error_Max-HopCount-Exceeded* (section 2.3.22)		
A_ARG_TYPE_Interfaces	*O*	**string**	*AllInterfaces*, *RequestInter-face* (section 2.3.23)		
A_ARG_TYPE_InterfaceResetStatus	*O*	**string**	*Success*, *Error* (section 2.3.24)		
A_ARG_TYPE_LogURI	*O*	**uri**	(section 2.3.25)		
A_ARG_TYPE_LogURL	*O*	**uri**	(section 2.3.26)		
A_ARG_TYPE_LogLevel	*O*	**string**	*Emergency*, *Alert*, *Critical*, *Error*, *Warning*, *Notice*, *Informational*, *Debug* (section 2.3.26)		
A_ARG_TYPE_LogMaxSize	*O*	**ui4**	(section 2.3.28)		Bytes
Non-standard state variables implemented by an UPnP vendor go here.	*X*	*TBD*	*TBD*	*TBD*	*TBD*

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

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

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.tz** [, **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 2.3.9).
- 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. *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 2.3.12 for a general description of test IDs. See Figure 2-1 for the test state transition diagram.

2.3.4. *LogURIs*

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

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 2.3.25 for a description of log URIs.

2.3.5. *A_ARG_TYPE_Boolean*

A boolean argument.

2.3.6. *A_ARG_TYPE_String*

A string argument.

2.3.7. *A_ARG_TYPE_UShort*

An unsigned short (ui2) argument.

2.3.8. A_ARG_TYPE_UInt

An unsigned int (ui4) argument.

2.3.9. 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.

2.3.10. A_ARG_TYPE_MSecs

A time interval measured in milliseconds.

2.3.11. 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 2-2: allowedValueList for A_ARG_TYPE_RebootStatus

Value	Req. or Opt.	Description
RebootNow	*R*	The *Reboot()* request was accepted, nothing in the current *Parent Device* state precludes an immediate reboot, and the *Parent Device* will initiate the reboot immediately.
RebootLater	*R*	The *Reboot()* request was accepted, but the *Parent Device* is unable to reboot now, e.g. because it is currently providing an important service, and will reboot as soon as possible.

2.3.12. 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 re-used 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 2-1.

2.3.13. 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

Value	Req. or Opt.	Description
NSLookup	*R*	

Value	Req. or Opt.	Description
Ping	*R*	
SelfTest	*R*	
Traceroute	*R*	
Vendor-defined	*X*[1]	

[1] For every vendor-defined diagnostic test, there MUST be a corresponding *A_ARG_TYPE_TestType* allowed value.

2.3.14. *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 2-1.

Table 2-4: allowedValueList for *A_ARG_TYPE_TestState*

Value	Req. or Opt.	Description
Requested	*R*	
InProgress	*R*	
Canceled	*R*	
Completed	*R*	

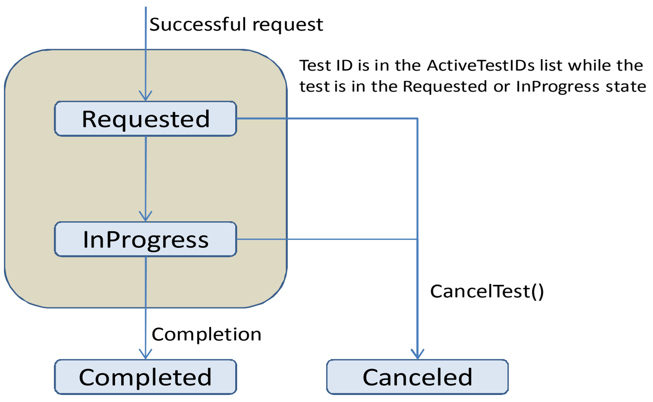
Successful request
Test ID is in the ActiveTestIDs list while the test is in the Requested or InProgress state
Requested
InProgress
CancelTest()
Completion
Completed
Canceled

Figure 2-1: Test State Transition Diagram

2.3.15. *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.

2.3.16. *A_ARG_TYPE_Host*

An IPv4/IPv6 address, DNS name, or empty string.

2.3.17. *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*).

2.3.18. *A_ARG_TYPE_HostName*

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

2.3.19. *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 2-5.

Table 2-5: allowedValueList for *A_ARG_TYPE_PingStatus*

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

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

2.3.20. *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 2-6.

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

Table 2-6: allowedValueList for *A_ARG_TYPE_NSLookupStatus*

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

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

2.3.21. *A_ARG_TYPE_NSLookupResult*

An XML document containing the result of a DNS lookup test. Figure 2-2 illustrates the XML Schema definition, which is given below, followed by an example XML document.

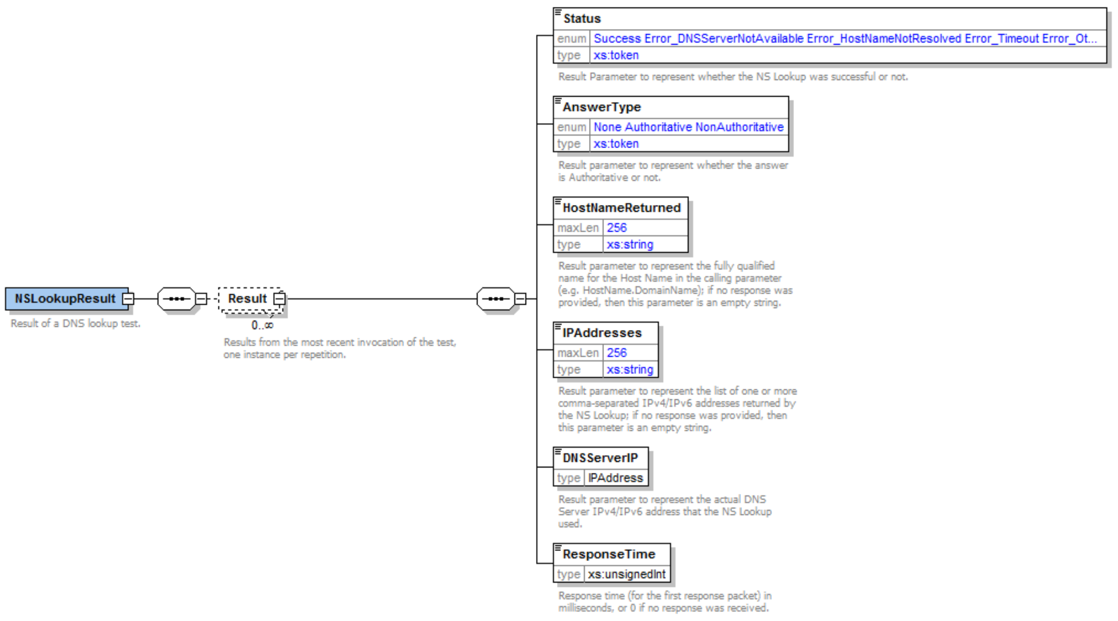
NSLookupResult
Result of a DNS lookup test.
Result
0..∞
Results from the most recent invocation of the test, one instance per repetition.
Status
enum Success Error_DNSServerNotAvailable Error_HostNameNotResolved Error_Timeout Error_Ot...
type xs:token
Result Parameter to represent whether the NS Lookup was successful or not.
AnswerType
enum None Authoritative NonAuthoritative
type xs:token
Result parameter to represent whether the answer is Authoritative or not.
HostNameReturned
maxLen 256
type xs:string
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.
IPAddresses
maxLen 256
type xs:string
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.
DNSServerIP
type IPAddress
Result parameter to represent the actual DNS Server IPv4/IPv6 address that the NS Lookup used.
ResponseTime
type xs:unsignedInt
Response time (for the first response packet) in milliseconds, or 0 if no response was received.

Figure 2-2: NSLookupResult XML Schema Diagram

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:bmsnsl="urn:schemas-upnp-org:dm:bms:nsl"
targetNamespace="urn:schemas-upnp-org:dm:bms:nsl"
elementFormDefault="unqualified" attributeFormDefault="unqualified"
version="1-yyyymmdd">
  <xs:simpleType name="IPAddress">
    <xs:annotation>
      <xs:documentation>IPv4/IPv6 address (currently just a
placeholder).</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:element name="NSLookupResult">
    <xs:annotation>
      <xs:documentation>Result of a DNS lookup test.</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <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: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="bmsnsl: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:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

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

2.3.22. *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 2-7.

Table 2-7: allowedValueList for *A_ARG_TYPE_TracerouteStatus*

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

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

2.3.23. *A_ARG_TYPE_Interfaces*

Indicates which IP interfaces should be reset by an interface reset test. Allowed values are listed in Table 2-8

Table 2-8: allowedValueList for *A_ARG_TYPE_Interfaces*

Value	Req. or Opt.	Description
AllInterfaces	*R*	All IP interfaces
RequestInterface	*R*	The IP interface on which the action request was received is to be reset.
NorthboundInterfaces	*O*	Relevant only to an Internet Gateway Device (a router that is connected to the Internet); the IP interface or interfaces via which traffic can be sent to or received from the Internet.
Vendor-defined	*X*[1]	MUST be the string "X_UPNP_ORG_" foll-owed by the name by which an IP interface is known in the [CMS] data model, i.e. the value of the corresponding `/UPnP/DM/Config-uration/Network/IPInterface/-#/SystemName` parameter.

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

2.3.24. *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 2-9.

Table 2-9: allowedValueList for *A_ARG_TYPE_InterfaceResetStatus*

Value	Req. or Opt.	Description
Success	*R*	
Error_Other	*R*	
Vendor-defined	*X*[1]	

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

2.3.25. *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. "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.

2.3.26. 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. "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.

2.3.27. A_ARG_TYPE_LogLevel

Indicates the lowest logging level that is included in a given log. Allowed values are listed in Table 2-10. 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 2-10: allowedValueList for A_ARG_TYPE_LogLevel[1]

Value	Req. or Opt.	Description
Emergency	*R*	
Alert	*R*	
Critical	*R*	
Error	*R*	
Warning	*R*	
Notice	*R*	
Informational	*R*	
Debug	*R*	
Vendor-defined	*X*[2]	

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

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

2.3.28. 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. Note that this is a fixed capability, not a "high water mark".

2.4. Eventing and Moderation

Table 2-11: Event Moderation

Variable Name	Evented	Moderated Event	Max Event Rate[1]	Logical Combination	Min Delta per Event[2]
DeviceStatus	*Yes*	*Yes*	1.0 second		

Variable Name	Evented	Moderated Event	Max Event Rate[1]	Logical Combination	Min Delta per Event[2]
SequenceMode	*Yes*	*No*			
ActiveTestIDs	*Yes*	*Yes*	0.2 seconds		
LogURIs	*Yes*	*No*			
Non-standard state variables implemented by an UPnP vendor go here.	*TBD*	*TBD*	*TBD*	*TBD*	*TBD*

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

2.4.1. SSDP 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 2-12).

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

Table 2-12: Allowed Values for *Announcement.dm.upnp.org field-value*

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

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

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

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

- The mechanism MUST only be used to announce important *Parent Device* state information that cannot reasonably be sent using GENA eventing.

- Each *field-value* corresponds to a piece of Boolean internal state information. It MUST be included in every *Parent Device* SSDP message (and SHOULD be included in every SSDP message for devices / services embedded within the *Parent Device*) if and only if the corresponding internal state is "**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.

2.5. Actions

Table 2-13: Actions

Name	Req. or Opt. [1]	Control Point R/O
Reboot()	*O*	*O*
BaselineReset()	*O*	*O*
GetDeviceStatus()	*R*	*R*
SetSequenceMode()	*O*	*O*
GetSequenceMode()	*O*	*O*
Ping()	*O*	*O*
GetPingResult()	*O*	*O*
NSLookup()	*O*	*O*
GetNSLookupResult()	*O*	*O*
Traceroute()	*O*	*O*
GetTracerouteResult()	*O*	*O*
InterfaceReset()	*O*	*O*
GetInterfaceResetResult()	*O*	*O*
SelfTest()	*O*	*O*
GetSelfTestResult()	*O*	*O*
GetActiveTestIDs()	*O*	*O*
GetTestInfo()	*O*	*O*
CancelTest()	*O*	*O*
GetLogURIs()	*O*	*O*
SetLogInfo()	*O*	*O*
GetLogInfo()	*O*	*O*
Non-standard actions implemented by an UPnP vendor go here.	*X*	*X*

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

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 2-14: Arguments for *Reboot()*

Argument	Direction	relatedStateVariable
RebootStatus	*OUT*	*A_ARG_TYPE_RebootStatus*

2.5.1.2. 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.

2.5.1.3. 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.

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.

2.5.1.4. Errors

Table 2-15: Error Codes for *Reboot()*

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

2.5.2. *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. 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.

2.5.2.3. 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.

2.5.2.4. Errors

Table 2-16: Error Codes for *BaselineReset()*

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

2.5.3. *GetDeviceStatus()*

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

2.5.3.1. Arguments

Table 2-17: Arguments for *GetDeviceStatus()*

Argument	Direction	relatedStateVariable
DeviceStatus	*OUT*	*DeviceStatus*

2.5.3.2. Dependency on State

None.

2.5.3.3. Effect on State

None.

2.5.3.4. Errors

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

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

2.5.4. *SetSequenceMode()*

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

- **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 2-19: Arguments for *SetSequenceMode()*

Argument	Direction	relatedStateVariable
NewSequenceMode	*IN*	*SequenceMode*

Argument	Direction	relatedStateVariable
OldSequenceMode	*OUT*	*SequenceMode*

2.5.4.2. Dependency on State

None.

2.5.4.3. Effect on State

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

2.5.4.4. Errors

Table 2-20: Error Codes for *SetSequenceMode()*

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

2.5.5. *GetSequenceMode()*

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

2.5.5.1. Arguments

Table 2-21: Arguments for *GetSequenceMode()*

Argument	Direction	relatedStateVariable
SequenceMode	*OUT*	*SequenceMode*

2.5.5.2. Dependency on State

None.

2.5.5.3. Effect on State

None.

2.5.5.4. Errors

Table 2-22: Error Codes for *GetSequenceMode()*

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

2.5.6. *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 maximum length of time (in milliseconds) to wait for each response before sending the next packet. A value of zero requests use of an implementation-chosen 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 2-23: Arguments for *Ping()*

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

2.5.6.2. Dependency on State

None.

2.5.6.3. Effect on State

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

2.5.6.4. Errors

Table 2-24: Error Codes for *Ping()*

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

2.5.7. *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 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 NumberOf-Repetitions).
- **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.

2.5.7.1. Arguments

Table 2-25: Arguments for *GetPingResult()*

Argument	Direction	relatedStateVariable
TestID	*IN*	*A_ARG_TYPE_TestID*
Status	*OUT*	*A_ARG_TYPE_PingStatus*
AdditionalInfo	*OUT*	*A_ARG_TYPE_String*
SuccessCount	*OUT*	*A_ARG_TYPE_UInt*
FailureCount	*OUT*	*A_ARG_TYPE_UInt*
AverageResponseTime	*OUT*	*A_ARG_TYPE_MSecs*
MinimumResponseTime	*OUT*	*A_ARG_TYPE_MSecs*
MaximumResponseTime	*OUT*	*A_ARG_TYPE_MSecs*

2.5.7.2. Dependency on State

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

2.5.7.3. Effect on State

None.

2.5.7.4. Errors

Table 2-26: Error Codes for *GetPingResult()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
706	No Such Test	No test with the specified TestID was found.
707	Wrong Test Type	TestID is valid but refers to a different test type.

errorCode	errorDescription	Description
708	Invalid Test State	The TestID is valid but test results are not available.

2.5.8. *NSLookup()*

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

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

- **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 2-27: Arguments for *NSLookup()*

Argument	Direction	relatedStateVariable
HostName	*IN*	*A_ARG_TYPE_HostName*
DNSServer	*IN*	*A_ARG_TYPE_Host*
NumberOfRepetitions	*IN*	*A_ARG_TYPE_UInt*
Timeout	*IN*	*A_ARG_TYPE_MSecs*
TestID	*OUT*	*A_ARG_TYPE_TestID*

2.5.8.2. Dependency on State

None.

2.5.8.3. Effect on State

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

2.5.8.4. Errors

Table 2-28: Error Codes for *NSLookup()*

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

errorCode	errorDescription	Description
703	Test Already Active	A test of this type is already active (and the implementation doesn't support multiple active instances of a given test type).
704	Capabilities Preclude Test	Test arguments are individually valid but, taken together, describe a test that is beyond the service's capabilities.
705	State Precludes Test	Service state precludes performing this test.

2.5.9. *GetNSLookupResult()*

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

- **Status**: indicates the overall success or failure of the test (if the test failed, the values of the remaining output arguments 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.

2.5.9.1. Arguments

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

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

2.5.9.2. Dependency on State

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

2.5.9.3. Effect on State

None.

2.5.9.4. Errors

Table 2-30: Error Codes for *GetNSLookupResult()*

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

errorCode	errorDescription	Description
706	No Such Test	No test with the specified TestID was found.
707	Wrong Test Type	TestID is valid but refers to a different test type.
708	Invalid Test State	The TestID is valid but test results are not available.

2.5.10. Traceroute()

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

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

- **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 2-31: Arguments for Traceroute()

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

2.5.10.2.Dependency on State

None.

2.5.10.3.Effect on State

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

2.5.10.4.Errors

Table 2-32: Error Codes for Traceroute()

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

2.5.11. *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 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 2-33: Arguments for *GetTracerouteResult()*

Argument	Direction	relatedStateVariable
TestID	*IN*	*A_ARG_TYPE_TestID*
Status	*OUT*	*A_ARG_TYPE_Traceroute-Status*
AdditionalInfo	*OUT*	*A_ARG_TYPE_String*
ResponseTime	*OUT*	*A_ARG_TYPE_MSecs*
HopHosts	*OUT*	*A_ARG_TYPE_Hosts*

2.5.11.2.Dependency on State

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

2.5.11.3.Effect on State

None.

2.5.11.4.Errors

Table 2-34: Error Codes for *GetTracerouteResult()*

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

2.5.12. *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:

- **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.

2.5.12.1.Arguments

Table 2-35: Arguments for *InterfaceReset()*

Argument	Direction	relatedStateVariable
Interfaces	*IN*	*A_ARG_TYPE_Interfaces*
TestID	*OUT*	*A_ARG_TYPE_TestID*

2.5.12.2.Dependency on State

None.

2.5.12.3.Effect on State

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

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.

2.5.12.4.Errors

Table 2-36: Error Codes for *InterfaceReset()*

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

2.5.13. *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 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.

2.5.13.1. Arguments

Table 2-37: Arguments for *GetInterfaceResetResult()*

Argument	Direction	relatedStateVariable
TestID	*IN*	*A_ARG_TYPE_TestID*
Status	*OUT*	*A_ARG_TYPE_Interface-ResetStatus*
AdditionalInfo	*OUT*	*A_ARG_TYPE_String*
NumberOfSuccesses	*OUT*	*A_ARG_TYPE_UShort*
NumberOfFailures	*OUT*	*A_ARG_TYPE_UShort*

2.5.13.2.Dependency on State

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

2.5.13.3.Effect on State

None.

2.5.13.4.Errors

Table 2-38: Error Codes for *GetInterfaceResetResult()*

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

2.5.14. *SelfTest()*

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

2.5.14.1.Arguments

Table 2-39: Arguments for *SelfTest()*

Argument	Direction	relatedStateVariable
TestID	*OUT*	*A_ARG_TYPE_TestID*

2.5.14.2.Dependency on State

None.

2.5.14.3.Effect on State

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

2.5.14.4.Errors

Table 2-40: Error Codes for *SelfTest()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
703	Test Already Active	A test of this type is already active (and the implementation doesn't support multiple active instances of a given test type).
705	State Precludes Test	Service state precludes performing this test.

2.5.15. 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.

2.5.15.1.Arguments

Table 2-41: Arguments for GetSelfTestResult()

Argument	Direction	relatedStateVariable
TestID	*IN*	*A_ARG_TYPE_TestID*
Status	*OUT*	*A_ARG_TYPE_Boolean*
AdditionalInfo	*OUT*	*A_ARG_TYPE_String*

2.5.15.2.Dependency on State

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

2.5.15.3.Effect on State

None.

2.5.15.4.Errors

Table 2-42: Error Codes for GetSelfTestResult()

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

2.5.16. 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.

2.5.16.1.Arguments

Table 2-43: Arguments for GetActiveTestIDs()

Argument	Direction	relatedStateVariable
TestIDs	*OUT*	*ActiveTestIDs*

2.5.16.2.Dependency on State

None.

2.5.16.3.Effect on State

None.

2.5.16.4.Errors

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

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

2.5.17. *GetTestInfo()*

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

2.5.17.1. Arguments

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

Argument	Direction	relatedStateVariable
TestID	*IN*	*A_ARG_TYPE_TestID*
Type	*OUT*	*A_ARG_TYPE_TestType*
State	*OUT*	*A_ARG_TYPE_TestState*

2.5.17.2.Dependency on State

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

2.5.17.3.Effect on State

None.

2.5.17.4.Errors

Table 2-46: Error Codes for *GetTestInfo()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
706	No Such Test	No test with the specified TestID was found.

2.5.18. *CancelTest()*

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

2.5.18.1. Arguments

Table 2-47: Arguments for *CancelTest()*

Argument	Direction	relatedStateVariable
TestID	*IN*	*A_ARG_TYPE_TestID*

2.5.18.2.Dependency on State

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

2.5.18.3.Effect on State

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

2.5.18.4.Errors

Table 2-48: Error Codes for *CancelTest()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
706	No Such Test	No test with the specified TestID was found.
709	State Precludes Cancel	The TestID is valid but the test can't be canceled.

2.5.19. *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.

2.5.19.1. Arguments

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

Argument	Direction	relatedStateVariable
LogURIs	*OUT*	*LogURIs*

2.5.19.2.Dependency on State

None.

2.5.19.3.Effect on State

None.

2.5.19.4.Errors

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

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.

errorCode	errorDescription	Description
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.

2.5.20. SetLogInfo()

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

2.5.20.1.Arguments

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

Argument	Direction	relatedStateVariable
LogURI	*IN*	*A_ARG_TYPE_LogURI*
Enabled	*IN*	*A_ARG_TYPE_Boolean*
LogLevel	*IN*	*A_ARG_TYPE_LogLevel*

2.5.20.2.Dependency on State

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

2.5.20.3.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.

2.5.20.4.Errors

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

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
710	No Such Log	No log with the specified LogURI was found.
711	Log Not Configurable	Log doesn't permit enable/disable and/or log level to be changed.

2.5.21. 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 2.3.28).

2.5.21.1.Arguments

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

Argument	Direction	relatedStateVariable
LogURI	*IN*	*A_ARG_TYPE_LogURI*
Configurable	*OUT*	*A_ARG_TYPE_Boolean*
Enabled	*OUT*	*A_ARG_TYPE_Boolean*
LogLevel	*OUT*	*A_ARG_TYPE_LogLevel*
LogURL	*OUT*	*A_ARG_TYPE_LogURL*
MaxSize	*OUT*	*A_ARG_TYPE_LogMaxSize*
LastChange	*OUT*	*A_ARG_TYPE_DateTime*

2.5.21.2. Dependency on State

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

2.5.21.3. Effect on State

None.

2.5.21.4. Errors

Table 2-54: Error Codes for *GetLogInfo()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
710	No Such Log	No log with the specified LogURI was found.

2.5.22. 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 2-55: Common Error Codes

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
700		Reserved for future extensions.
701	Interface Not Found	The requested IP interface was not found.
702	Interface Not Resettable	One or more of the requested IP interfaces has a static address and cannot be reset (InterfaceReset).
703	Test Already Active	A test of this type is already active (and the implementation doesn’t support multiple active instances of a given test type).

errorCode	errorDescription	Description
704	Capabilities Preclude Test	Test arguments are individually valid but, taken together, describe a test that is beyond the service's capabilities.
705	State Precludes Test	Service state precludes performing this test.
706	No Such Test	No test with the specified TestID was found.
707	Wrong Test Type	TestID is valid but refers to a different test type.
708	Invalid Test State	The TestID is valid but test results are not available.
709	State Precludes Cancel	The TestID is valid but the test can't be canceled.
710	No Such Log	No log with the specified LogURI was found.
711	Log Not Configurable	Log doesn't permit enable/disable and/or log level to be changed.
800-899	*TBD*	*(Specified by UPnP vendor.)*

2.6. Theory of Operation

This non-normative (informative) section walks through several scenarios to illustrate the various actions supported by the BasicManagement:1 service.

2.6.1. Assumptions

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

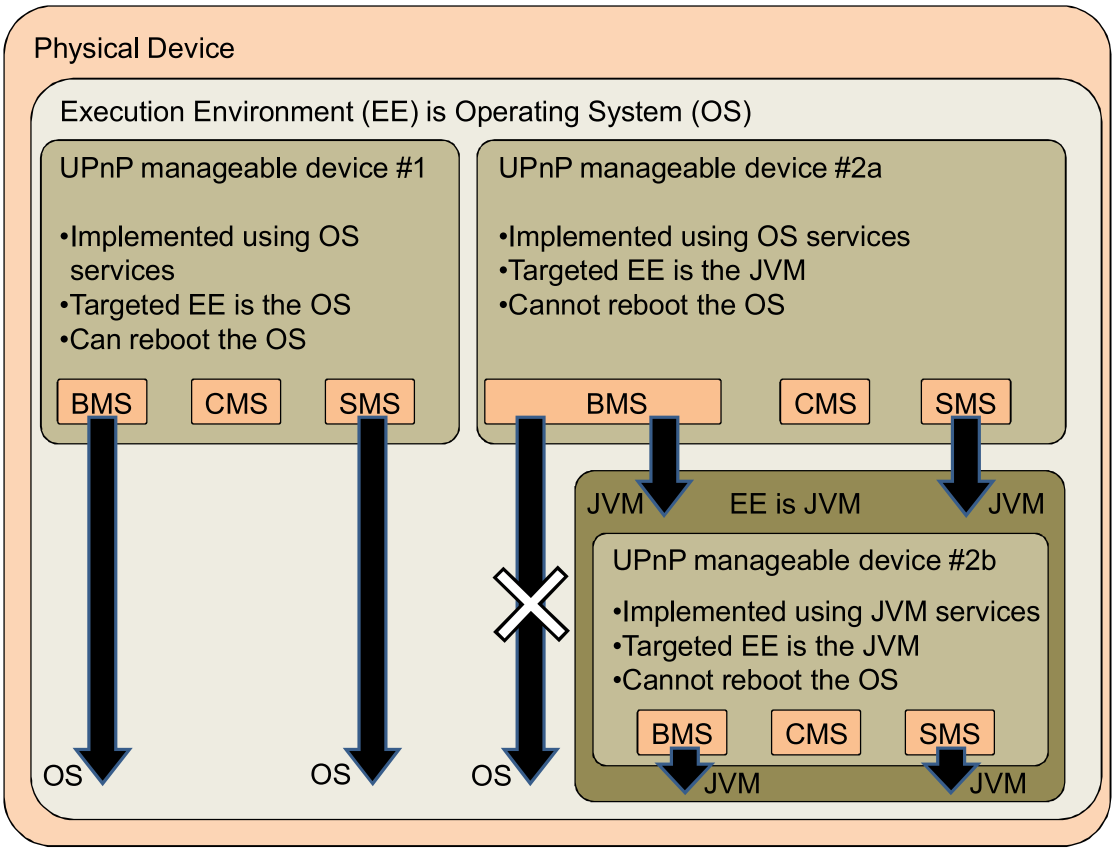
Physical Device
Execution Environment (EE) is Operating System (OS)
UPnP manageable device #1
•Implemented using OS services
•Targeted EE is the OS
•Can reboot the OS
BMS
CMS
SMS
UPnP manageable device #2a
•Implemented using OS services
•Targeted EE is the JVM
•Cannot reboot the OS
BMS
CMS
SMS
JVM
EE is JVM
JVM
UPnP manageable device #2b
•Implemented using JVM services
•Targeted EE is the JVM
•Cannot reboot the OS
BMS
CMS
SMS
OS
OS
OS
JVM
JVM

Figure 2-3: Example *Parent Devices*

2.6.1.1. 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.

2.6.1.2. 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.

2.6.1.3. 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.

2.6.2. Rebooting the *Parent Device*

The *Reboot()* action (section 2.5.1) is required, but the implementation has some leeway in deciding whether to accept the request.

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*.

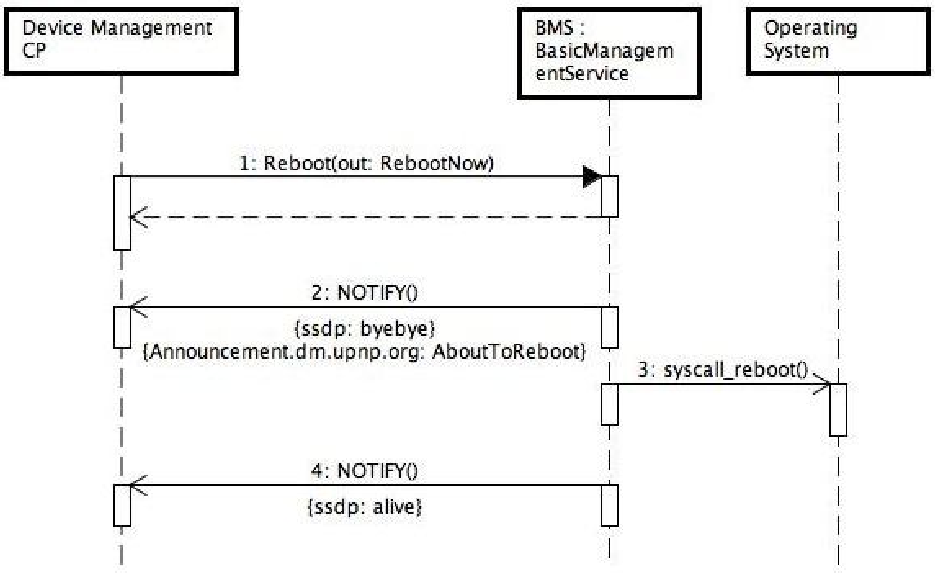

Figure 2-4: *RebootNow* Example

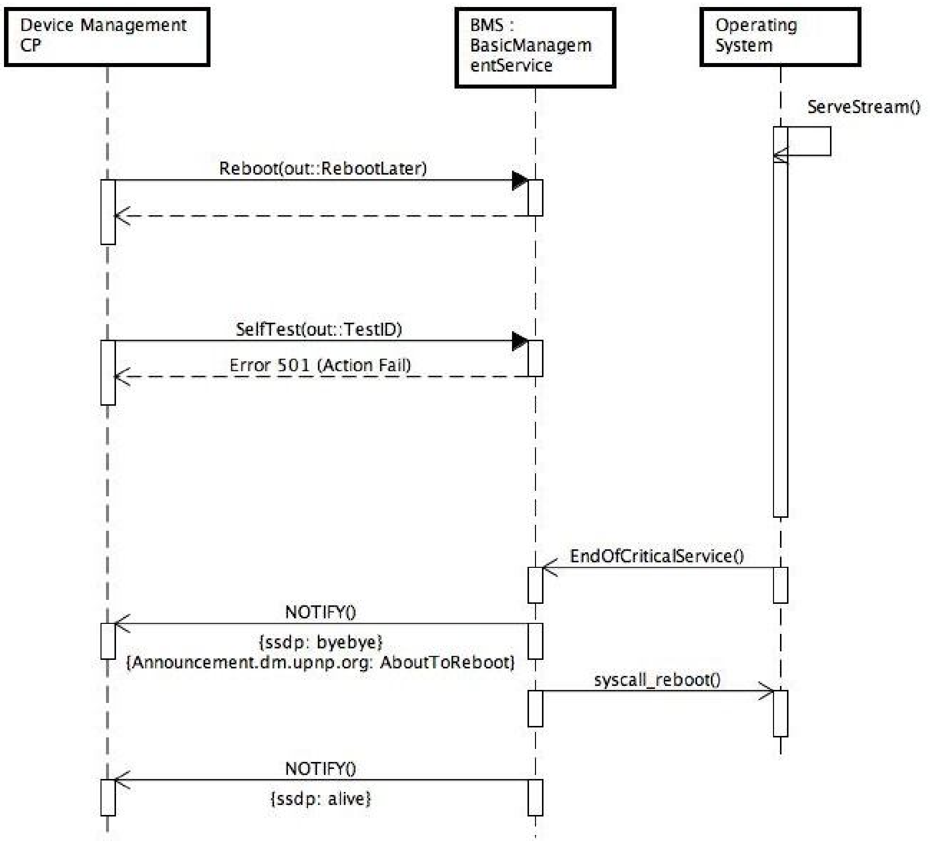

Figure 2-5: *RebootLater* Example

2.6.3. Resetting the *Parent Device*

The *BaselineReset()* action (section 2.5.2) is optional. If supported, it 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:

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

~~*ANNOUNCEMENT.DM.UPNP.ORG: AboutToReboot, AboutToBaselineReset*~~

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

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

- Reboot, as described in section 2.6.2.
- Persistent settings revert to their baseline values.

2.6.4. Using Sequence Mode

The *SetSequenceMode()* and *GetSequenceMode()* actions (Sections 2.5.4 and 2.5.5) control 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**".

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*.

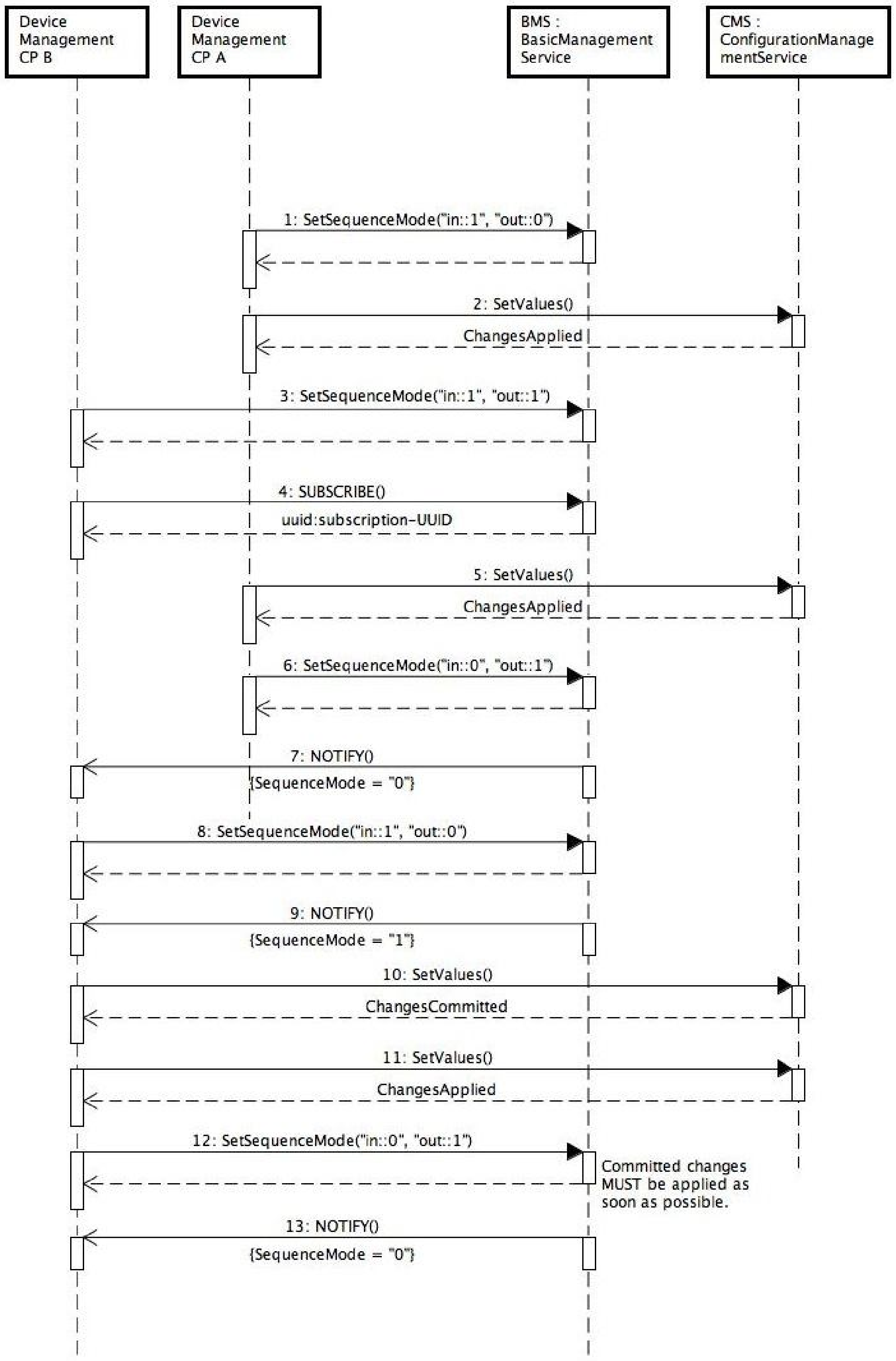
Device Management CP B
Device Management CP A
BMS : BasicManagement Service
CMS : ConfigurationManagementService
1: SetSequenceMode("in::1", "out::0")
2: SetValues()
ChangesApplied
3: SetSequenceMode("in::1", "out::1")
4: SUBSCRIBE()
uuid:subscription-UUID
5: SetValues()
ChangesApplied
6: SetSequenceMode("in::0", "out::1")
7: NOTIFY()
{SequenceMode = "0"}
8: SetSequenceMode("in::1", "out::0")
9: NOTIFY()
{SequenceMode = "1"}
10: SetValues()
ChangesCommitted
11: SetValues()
ChangesApplied
12: SetSequenceMode("in::0", "out::1")
Committed changes MUST be applied as soon as possible.
13: NOTIFY()
{SequenceMode = "0"}

Figure 2-6: *SequenceMode* Example

2.6.5. Running a Ping Test

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:1* events and discovers that *ActiveTestIDs* is "", indicating that no tests are currently active.
- Control point calls *Ping()*. For example:
  - *Ping("www.google.com", 0, 0, 0, 0)* : the four zeroes are (respectively) the number of repetitions (defaulted), the timeout (defaulted), the data block size (defaulted) and the DSCP value (best effort). The defaults are implementation-dependent.
  - *Ping("www.google.com", 10, 1000, 32, 16)* : 10 repetitions, 1000 millisecond timeout, 32 byte data block, DSCP of 16.
- Control point receives test ID (42 for example) in the *Ping()* response. It also discovers, via an event, that *ActiveTestIDs* is now "42".
- *Parent Device* performs the test and, on completion, removes test ID 42 from *ActiveTestIDs*.
- Control point discovers, via an event, that *ActiveTestIDs* is "" and knows that the test is complete.
- Control point calls *GetPingResult(42)* to retrieve the test results. For example:
  - *GetPingResult(42)→("Success", "", 9, 1, 45, 40, 50)* indicates that the ping test was successful, no additional information string was returned, and 9/10 pings succeeded with a mean/min/max response times (for the successful pings) of 45ms/40ms/50ms.
  - *GetPingResult(42 →("Error_CannotResolveHostName", "Timeout", 0, 0, 0, 0, 0)* indicates that the ping test failed because the host name couldn't be resolved. The free-format additional info indicates a timeout, and the values of the remaining output arguments are irrelevant (because the test failed).

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

2.6.6. Running an NSLookup Test

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:1* events and discovers that *ActiveTestIDs* is "", indicating that no tests are currently active.
- Control point calls *NSLookup()*. For example:
  - *NSLookup("www.myserver.com", "", 0, 0)* : the empty string indicates that the default DNS server will be used, and the two zeroes are (respectively) the number of repetitions (defaulted) and the timeout (defaulted). The defaults are implementation-dependent.
  - *Ping("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 2.3.21 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.

2.6.7. Running a Traceroute Test

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:1* events and discovers that *ActiveTestIDs* is “”, indicating that no tests are currently active.
- Control point calls *Traceroute()*. For example:
  - *Traceroute(“www.myserver.com”, 0, 0, 0, 0)* : the four zeroes are (respectively) the timeout (defaulted), the data block size (defaulted), the maximum hop count (defaulted) and the DSCP value (best effort). The defaults are implementation-dependent.
  - *Traceroute(“www.myserver.com”, 1000, 32, 20, 16)* : 1000 millisecond timeout, 32 byte data block, maximum hop count of 20, DSCP of 16.
- Control point receives test ID (44 for example) in the *Traceroute()* response. It also discovers, via an event, that *ActiveTestIDs* is now “44”.
- *Parent Device* performs the test and, on completion, removes test ID 44 from *ActiveTestIDs*.
- Control point discovers, via an event, that *ActiveTestIDs* is “” and knows that the test is complete.
- Control point calls *GetTracerouteResult(44)* to retrieve the test results. For example:
  - *GetTracerouteResult(44)→ (“Success”, “”, 888, “1.2.3.4,2.3.4.5,,4.5.6.7”)* indicates that the trace-route test was successful, no additional information string was returned, the average round-trip time to *www.myserver.com* was 888 milliseconds, and there were four hops to *www.myserver.com*. The third entry in the list of hops is empty, indicating that no replies were received from it. The final entry will be *www.myserver.com*’s IP address.
  - *GetTracerouteResult(44) → (“Error_MaxHopCountExceeded”, “Timeout”, 0, “”)* indicates that the trace-route test failed because the number of hops to *www.mysever.com* is more than the supplied hop count. The free-format additional info indicates a timeout, and the values of the remaining output arguments are irrelevant (because the test failed).

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

2.6.8. Running an InterfaceReset Test

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

- Control point subscribes to *BasicManagement:1* events and discovers that *ActiveTestIDs* is "", indicating that no tests are currently active.
- Control point calls *InterfaceReset()*. For example:
  - *InterfaceReset("AllInterfaces")* : reset all IP interfaces.
  - *InterfaceReset("RequestInterface")* : reset the IP interface on which the action request was received.
  - *InterfaceReset("X_UPNP_ORG_lan")* : reset the IP interface whose system name is *lan*. If no such interface exists, the request will be rejected with a 701 (Interface Not Found) error.
- Control point receives test ID (45 for example) in the *InterfaceReset()* response. It also discovers, via an event, that *ActiveTestIDs* is "45".
- *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 45 needs to persist across this reset. On completion, test ID 45 is removed from *ActiveTestIDs*.
- Control point discovers, via an event, that *ActiveTestIDs* is "" and knows that the test is complete, i.e., test ID 45 has been removed because of the test completion.
- Control point calls *GetInterfaceResetResult(45)* and retrieves the test results. For example:
  - *GetInterfaceResetResult(45)→ ("Success", "", 1, 0)* indicates that the interface reset test was successful, no additional information string was returned, one IP interface was successfully reset, and no IP interfaces could not be reset.
  - *GetInterfaceResetResult(45)→ ("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(45)*, which returns the test type (*InterfaceReset*) and the test state (*Requested*, *InProgress*, *Canceled*, *Completed*). Once it changes to *Completed*, *GetInterfaceResetResult(45)* can be called to return the results.

2.6.9. Running a Self Test

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:1* events and discovers that *ActiveTestIDs* is "", indicating that no tests are currently active.
- Control point calls *SelfTest()*.
- Control point receives test ID (46 for example) in the *SelfTest()* response. It also discovers, via an event, that *ActiveTestIDs* is "46".
- *Parent Device* performs the test and, on completion, removes test ID 46 from *ActiveTestIDs*.

- Control point discovers, via an event, that *ActiveTestIDs* is "" and knows that the test is complete, i.e., test ID 46 has been removed because of the test completion.
- Control point calls *GetSelfTestResult(46)* and retrieves the test results. For example:
  - *GetSelfTestResult(46)→ (1, "")* indicates that the self test was successful, but no additional information string was returned.
  - *GetSelfTestResult(46)→ (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(46)*, which returns the test type (*SelfTest*) and the test state (*Requested*, *InProgress*, *Canceled*, *Completed*). Once it changes to *Completed*, *GetSelfTestResult(46)* can be called to return the results.

2.6.10. Manipulating Logs

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

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

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

- *GetLogInfo("urn:example-com:device-log") → (1, 1, "Error", "http://192.168.1.254/device-log", 0, 2009-06-15T14:00:00)* indicates that the specified log is configurable, is enabled, its current log level (*Error*), its log URL (*http://192.168.1.254/device-log*), its maximum size (0 means unknown), and the time at which it last changed.
- *GetLogInfo("urn:example-com:jvm-log") → (0, 1, "Info", "http://192.168.1.254/jvm-log", 100000, 2009-06-15T14:00:00)* indicates that the specified log is not configurable, is enabled, its current log level (*Info*), its log URL (*http://192.168.1.254/jvm-log*), its maximum size (100000 bytes), and the time at which it last changed.
- *SetLogInfo("urn:example-com:device-log", 1, Info)* changes the log level from *Error* to *Info*.
- *SetLogInfo("urn:example-com:jvm-log", 1, Info)* fails with a 711 (Log Not Configurable) error.

3. XML Service Description

```
<?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>
    <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_UShort</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>
```

```
<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>
```

```
<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>
        <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>
      </argumentList>
    </action>
    <action>
      <name>GetTracerouteResult</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_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>
      </argumentList>
    </action>
    <action>
      <name>InterfaceReset</name>
      <argumentList>
        <argument>
```

```
          <name>Interfaces</name>
          <direction>in</direction>

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

<relatedStateVariable>A_ARG_TYPE_TestID</relatedStateVariable>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>GetInterfaceResetResult</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_InterfaceResetStatus</relatedStateVari
able>
        </argument>
        <argument>
          <name>AdditionalInfo</name>
          <direction>out</direction>

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

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

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

<relatedStateVariable>A_ARG_TYPE_TestID</relatedStateVariable>
```

```
        </argument>
      </argumentList>
    </action>
    <action>
      <name>GetSelfTestResult</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_Boolean</relatedStateVariable>
        </argument>
        <argument>
          <name>AdditionalInfo</name>
          <direction>out</direction>

<relatedStateVariable>A_ARG_TYPE_String</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>LogURIs</relatedStateVariable>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>SetLogInfo</name>
      <argumentList>
        <argument>
          <name>LogURI</name>
          <direction>in</direction>

<relatedStateVariable>A_ARG_TYPE_LogURI</relatedStateVariable>
        </argument>
        <argument>
          <name>Enabled</name>
          <direction>in</direction>

<relatedStateVariable>A_ARG_TYPE_Boolean</relatedStateVariable>
        </argument>
        <argument>
          <name>LogLevel</name>
          <direction>in</direction>

<relatedStateVariable>A_ARG_TYPE_LogLevel</relatedStateVariable>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>GetLogInfo</name>
      <argumentList>
        <argument>
          <name>LogURI</name>
          <direction>in</direction>

<relatedStateVariable>A_ARG_TYPE_LogURI</relatedStateVariable>
        </argument>
        <argument>
          <name>Configurable</name>
          <direction>out</direction>

<relatedStateVariable>A_ARG_TYPE_Boolean</relatedStateVariable>
        </argument>
        <argument>
          <name>Enabled</name>
          <direction>out</direction>

<relatedStateVariable>A_ARG_TYPE_Boolean</relatedStateVariable>
        </argument>
        <argument>
          <name>LogLevel</name>
          <direction>out</direction>

<relatedStateVariable>A_ARG_TYPE_LogLevel</relatedStateVariable>
        </argument>
        <argument>
          <name>LogURL</name>
          <direction>out</direction>

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

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

<relatedStateVariable>A_ARG_TYPE_DateTime</relatedStateVariable>
        </argument>
      </argumentList>
    </action>
  </actionList>
  <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>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>
    <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>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_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_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>
        <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>
  </serviceStateTable>
</scpd>
```